Partage d'interactions en environnements virtuels : de nouvelles techniques collaboratives basées sur un protocole de dialogue générique

Laurent Aguerreche

Thèse

THÈSE INSA Rennes
sous le sceau de l'Université européenne de Bretagne
pour obtenir le titre de
DOCTEUR DE L'INSA DE RENNES
Spécialité : Informatique

présentée par
Laurent Aguerreche
ÉCOLE DOCTORALE : MATISSE
LABORATOIRE : IRISA

Partage d'interactions en environnements virtuels : de nouvelles techniques collaboratives basées sur un protocole de dialogue générique

Thèse soutenue le 04.06.2010
devant le jury composé de :

Jean-Louis Pazat
Professeur des universités — INSA de Rennes / *président*
Sabine Coquillart
Directrice de recherche — INRIA Grenoble - Rhône-Alpes / *rapporteur*
Indira Mouttapa Thouvenin
Enseignant-Chercheur HDR — Université de Technologie Compiègne / *rapporteur*
Martin Hachet
Chargé de recherche — INRIA Bordeaux - Sud-Ouest / *examinateur*
Thierry Duval
Maître de conférences — Université de Rennes 1 / Co-encadrant de thèse
Bruno Arnaldi
Professeur des universités — INSA de Rennes / Directeur de thèse

Partage d'interactions en environnements virtuels : de nouvelles techniques collaboratives basées sur un protocole de dialogue générique

Laurent Aguerreche

En partenariat avec

La réalité, quelle qu'elle soit, est bien plus belle que l'illusion...

Sacha Guitry,
extrait de la pièce de théatre *Mon père avait raison.*

Remerciements

Je tiens tout d'abord à remercier Bruno Arnaldi, professeur à l'INSA de Rennes, pour avoir accepté de me faire confiance en encadrant cette thèse. Durant toute cette période, tu as toujours été présent pour me conseiller sur mon travail et m'aider à prendre du recul. Je remercie ensuite Thierry Duval pour avoir également su me faire confiance et m'avoir supporté durant ces trois années même s'il semblerait que ce n'eut pas toujours été facile. Merci surtout pour ta gentillesse et ton implication.

Je remercie les deux rapporteurs de ce mémoire qui ont accepté de l'évaluer et de faire partie du jury de thèse : Sabine Coquillart, directrice de recherche à l'Inria Grenoble et Indira Mouttapa Thouvenin, enseignant-chercheur avec habilitation à diriger des recherches à l'université technique de Compiègne.

Je remercie les examinateurs du jury de thèse. Jean-Louis Pazat, professeur à l'INSA de Rennes, pour avoir accepté de présider ce jury. Martin Hachet, chargé de recherche à l'Inria Bordeaux, pour outre sa participation au jury, son support qui m'a aidé à obtenir la confiance de Bruno et Thierry.

Je remercie bien évidemment l'équipe Bunraku dont l'ambiance a toujours été très chaleureuse. D'abord, merci à Valérie Gouranton d'avoir râlé de concert avec Thierry et moi en partageant le même bureau pendant près d'un an et demi. Merci ensuite à Cédric Fleury de faire désormais de même mais sans Thierry. Merci d'ailleurs pour ton écoute et bon courage pour ta thèse. Merci à tous ceux qui m'ont aidé à m'occuper de près ou de loin du phœnix OpenMASK : Benoît Chanclou, dont la participation a été très importante lors de l'implémentation du protocole d'interaction, Xavier Larrodé, principalement pour son travail autour de Ogre3D, Michaël Rouillé (désolé de t'associer à cette plate-forme logicielle que tu aimes tant), pour ses avis éclairés sur le rendu 3D et l'architecture logicielle. Merci à Florian Nouviale de m'avoir épaulé depuis plusieurs mois en continuant à effectuer du support et des évolutions. Merci à Alain Chauffaut d'être la mémoire vivante !

Et pour continuer avec Bunraku, je remercie tout ceux qui ont été si agréables avec moi (dans le désordre) : Fabien (bonne arrivée à Bordeaux !), Loïc Magic T., Stéphanie, Coralie, Claudie (mais je ne te remercie pas pour ton gâteau au Mars^{TM}), Noémie (je n'oublie pas la formation bisounours avec toi), Anne-Hélène, Maud, Anatole (je suis toujours aussi surpris de voir tes capacités pour la relecture d'articles et les riches conseils que tu peux donner), Bart, les Yann/Ian, Aurélien, Laurent, Laurent (l'autre), Charles, Ludo, Gabriel, Tony, Jean, Jonathan, Vincent Toï-Toï, Vincent Boobs-Manager (et sa femme Loeiz), Léo, Séb (quand il réussit à être à l'Irisa), Nico P. (courage pour

la suite), le fameux Samuel, et puis tous ceux que j'ai oublié de citer ! D'un point de vue politique, je pense qu'il est important que je cite le précédent chef de Bunraku, donc merci à Stéphane de me montrer qu'il est une bonne chose de porter des couleurs, et l'actuel chef, donc merci à Georges pour avoir notamment une belle moustache.

Thanks to some people who spent only a few months in our team but are important to me. So thank you Raquel for speaking all the time with so much humor. And thank you Yun-Tai for many things.

À l'Irisa, j'ai aussi eu l'occasion de rencontrer des gens inoubliables dans d'autres équipes. Par exemple, un certain Gwénolé qui aime se déguiser en pirate.

Merci à tous ceux avec lesquels j'ai eu une collaboration étroite au cours du projet ANR Part@ge dont : Céline Chatelain, Lionel Dominjon, Alexandre Bouchet, Jérôme Ardouin, Samuel Degrande, François Vogel, Jean-Emmanuel Viallet.

Évidemment, je remercie mes amis de longue date pour être toujours là : Jérôme, Clément, Audrey, Arnaud, Aurélie. Et puis, il y a Docteur Fabrice Polnafa De(s)cle, Christophe Punky (un grand merci d'avoir toujours été si présent pour moi à Rennes), Julien P'tit Bouchon. Merci aussi aux ex ou actuels bretons qui comptent aujourd'hui pour moi : Pierco, Cécile, Antoine, Clément, Zoé et Sandrine.

Pour ma famille, je remercie ma cousine Delphine pour toute sa bonne humeur et sa gentillesse. À Hervé et toi, je vous souhaite beaucoup de bonheur avec la petite Tiphenn. Merci à sœurette de se soucier de moi. Merci à mes parents sans qui tout cela ne serait pas arrivé mais surtout de m'avoir donner tant de liberté.

Table des matières

Introduction

La réalité virtuelle trouve des champs d'applications de plus en plus larges qui s'étendent du domaine ludique aux activités professionnelles. Elle est notamment utilisée par des industriels de l'automobile, de l'aviation ou des transports. Ces industriels l'emploient généralement dans le but de réaliser des maquettes numériques, c'est-à-dire de créer une représentation d'objets sur lesquels travailler dans un environnement virtuel. Par exemple, il peut s'agir de représenter un futur modèle de voiture. Ces simulations tendent à se démocratiser parce que les ordinateurs actuels permettent de produire des environnements dont l'apparence et les comportements des objets qu'ils représentent se rapprochent de la réalité. Toutefois, le but de la réalité virtuelle n'est pas nécessairement d'imiter la réalité le plus fidèlement possible mais de proposer à l'utilisateur une représentation immersive en stimulant ses canaux sensori-moteurs.

L'usage de maquettes numériques dans l'industrie vise à donner l'impression à une personne qu'elle les manipule selon des contraintes similaires à celles de la réalité. À cette fin, la représentation d'objets est souvent effectuée à l'échelle 1:1. Tout en évitant de fabriquer des prototypes réels pour faire baisser les coûts de développement, il est ainsi possible de pratiquer diverses études afin de concevoir un nouveau produit. Prenons l'exemple d'une voiture. Des techniciens peuvent vérifier que des opérations de montage/démontage de pièces sont possibles et le resteront une fois le véhicule terminé. Des ingénieurs peuvent pratiquer des calculs de déformations physiques à la suite de chocs. Des ergonomes peuvent évaluer le tableau de bord pour l'accès aux commandes. En outre, certaines opérations peuvent nécessiter des actions à plusieurs personnes. Par exemple, un pare-brise de voiture peut être posé sur un véhicule par deux personnes.

Malgré l'existence de manipulations collaboratives dans le monde réel et l'imitation de la réalité qui peut être souhaitée dans les environnements virtuels, les applications de réalité virtuelle sont souvent limitées à un seul utilisateur. Par exemple, un utilisateur effectue généralement seul la pose d'un pare-brise virtuel de voiture. Par conséquent, des contraintes inhérentes au travail collaboratif du monde réel sont ignorées comme la difficulté de la coordination de mouvements entre deux personnes ou la difficulté à discuter des mêmes choses précisément.

À ce souhait de pouvoir interagir collaborativement peut s'ajouter celui de rendre possible des interactions multi-sites où les utilisateurs sont donc distants physiquement. Des personnes distantes ont alors la possibilité de se retrouver dans un environnement virtuel commun. La réalité virtuelle peut constituer un moyen de diminution des coûts relatifs à des déplacements de personnels. Aujourd'hui, il devient courant que des revues

de projets s'effectuent depuis plusieurs lieux géographiques. Dans le cas plus particulier des manipulations collaboratives d'objets virtuels, des actions depuis des lieux distants entraînent des difficultés. En effet, à celles du monde réel s'ajoutent celles provenant de l'environnement virtuel : des diminutions possibles du champ de vision, une qualité audio ne reproduisant pas la réalité, un sens du touché appauvri, voire inexistant. La conception d'environnements virtuels pour les interactions collaboratives doit prendre en compte ces limitations et essayer de les compenser.

Pour répondre aux besoins de collaboration distante et résoudre les difficultés inhérentes à la communication, cette thèse présente un travail portant sur la collaboration entre au moins deux utilisateurs amenés à interagir simultanément dans un environnement virtuel commun. Les actions d'un utilisateur peuvent donc être perçues au même moment par un autre. Par conséquent, les interactions collaboratives que nous allons étudier se placent principalement dans le cas où des utilisateurs, éventuellement distants, co-manipulent, c'est-à-dire qu'ils manipulent simultanément un même objet virtuel.

Le chapitre 1 présente un état de l'art des techniques d'interaction mono et multi utilisateurs en environnements 3D virtuels immersifs, ainsi que des fondations techniques employées : abstraction du matériel pour la réalité virtuelle, langages de description d'environnements virtuels, architectures réseaux et algorithmes réseau pour les plates-formes distribuées de réalité virtuelle.

Le chapitre 2 analyse l'existant en matière de techniques d'interaction collaboratives et de leur mise en œuvre sur des plates-formes de réalité virtuelle, puis propose des améliorations pour la mise en œuvre des techniques d'interaction et pour les interactions collaboratives.

Le chapitre 3 décrit un protocole d'échange entre outils d'interaction et objets interactifs. La réflexion autour de ce protocole nous conduira à expliquer sa mise en œuvre et à montrer la description de propriétés interactives d'objets interactifs grâce à une extension du langage COLLADA [COL][1].

Le chapitre 4 présente une nouvelle technique d'interaction collaborative dont l'une des particularités est d'utiliser 3 mains virtuelles.

Le chapitre 5 présente et évalue une nouvelle technique d'interaction basée sur une interface tangible dont la particularité est d'être reconfigurable. Deux instances de cette interface sont présentées. L'une est triangulaire afin de fournir 3 points de manipulation. L'autre comporte 4 points et est réalisée sous une forme plane et une forme non-plane.

[1] Dans ce manuscrit, les références portant sur des sites de l'internet sont rangées dans la partie *webographie* après la bibliographie.

Le chapitre 6 présente l'architecture de couplage employée entre la plate-forme de réalité virtuelle utilisée et un moteur physique. Ce couplage a servi de socle technique à la technique basée sur 3 mains virtuelles et à l'interface tangible reconfigurable.

Ce travail sur les interactions collaboratives dans des environnements virtuels 3D immersifs a été effectué dans le cadre du projet ANR Part@ge [Par] rassemblant plusieurs partenaires académiques (Insa de Rennes — porteur du projet —, Inria-Alcove, Inria-I3D, CNRS-LaBRI, CNRS-ISM, Esiea) et industriels (CEA-List, Clarté, Haption, 3DVIA Virtools, Sogitec, France Telecom R&D, Renault, Thalès). Pendant toute sa durée, de 2007 à 2010, Part@ge a travaillé sur les interactions collaboratives *via* plusieurs lots techniques qui portaient sur les modèles et objets pour les environnements virtuels collaboratifs (lot 1), sur la communication et la présence (lot 2), sur la collaboration avancée (lot 3), et sur l'intégration et le déploiement des résultats de ces lots (lot 4).

La thèse présentée se situe dans le cadre de plusieurs lots techniques du projet Part@ge. Ainsi, le chapitre 3 présente un travail réalisé dans le cadre du lot 1 de Part@ge. Le travail autour de COLLADA a été réalisé en partenariat avec Sogitec, Clarté, Inria-Alcove et France Telecom R&D. La définition du protocole d'interaction a été effectuée en collaboration avec Clarté, Inria-Alcove et France Telecom R&D. Le chapitre 4 et le chapitre 5 ont été réalisés dans le cadre des lots 1 et 4 de Part@ge.

Chapitre 1

État de l'art

1.1 Introduction

Cet état de l'art [ADA09b] présente tout d'abord des métaphores d'interaction pour un seul utilisateur. La plupart de ces métaphores ne peuvent être employées directement dans un environnement coopératif parce que les plates-formes logicielles ne sont généralement pas prévues pour gérer des accès concurrents à un objet et parce qu'un utilisateur n'a aucun moyen de savoir qu'il est en train d'interagir simultanément sur un objet avec quelqu'un d'autre. Certaines métaphores d'interaction peuvent cependant être rapidement rendues collaboratives si elles ne requièrent que l'ajout de retours d'informations aux utilisateurs. Par conséquent, cet état de l'art traite de la coopération selon un plan similaire à celui des interactions mono-utilisateur.

Outre les techniques d'interaction, nous abordons aussi les problèmes des architectures logicielles sous-jacentes. Nous traitons ainsi de l'abstraction du matériel de réalité virtuelle puis de la description d'environnements virtuels et des techniques d'interaction. Nous parlons également des architectures réseau employées et de leurs algorithmes de gestion de la synchronisation et du maintien de la cohérence.

1.2 Réalité virtuelle et interactions

Les définitions concernant la réalité virtuelle sont nombreuses et peuvent être très détaillées comme celle présente dans le traité de la réalité virtuelle [AFT03]. Nous choisissons ici une définition compacte qui est utilisée dans le cadre du projet ANR Part@ge [Par] : « la *réalité virtuelle* peut être définie comme un ensemble de techniques matérielles et logicielles destinées à permettre à un ou plusieurs utilisateurs d'interagir de la façon la plus naturelle possible avec des données numériques ressenties par le biais de canaux sensoriels ».

Les interactions entre des utilisateurs humains et des objets d'un monde virtuel — que nous nommerons parfois *objets virtuels* par raccourci dans cet état de l'art — se font au travers de techniques ou de métaphores d'interaction. Selon l'état de l'art du projet RNTL Perf-rv [per] : « une technique, ou paradigme, d'interaction désigne une

méthode, une stratégie, un scénario d'utilisation, permettant à l'utilisateur d'accomplir une tâche précise dans l'univers virtuel : naviguer, sélectionner des objets, manipuler des objets, contrôler l'application » ; « une métaphore d'interaction est utilisée par les fonctions de transfert. Une métaphore est la transposition d'un objet ou d'un concept réel utilisé pour faciliter l'interaction ».

Dans le cadre d'interactions, un utilisateur peut être remplacé par une entité logicielle. Pour cette raison, nous employons aussi le terme d'*interacteur* qui désigne une entité en train d'agir sur l'environnement virtuel.

Dans un environnement virtuel, un utilisateur effectue plusieurs types de tâches : navigation, sélection/manipulation d'objets, contrôle d'application [Han97]. Nous choisissons de nous intéresser à la sélection/manipulation d'objets. L'utilisateur interagit avec les éléments d'un environnement virtuel par le biais de techniques d'interaction. La première partie de la section suivante concerne les techniques d'interaction et est suivie d'une partie traitant de la perception du potentiel interactif des objets.

1.2.1 Techniques de sélection/manipulation

Nous allons présenter un panorama des principales techniques d'interaction rencontrées dans les environnements virtuels. Selon [PWBI98], elles peuvent être classées en deux grands groupes : d'un côté, les interactions *exocentriques*, ou encore du point de vue de Dieu[1], et de l'autre les interactions *égocentriques*.

1.2.1.1 Interactions exocentriques

Les interactions exocentriques placent un interacteur dans une position extérieure à l'environnement virtuel de sorte que l'environnement puisse être perçu dans son ensemble. Dans cette situation, l'interacteur n'est plus forcément contraint par la gravité ou encore par sa taille vis-à-vis des éléments de l'environnement virtuel.

Dans la technique dite du monde en miniature (WIM[2], [SCP95]), l'environnement est représenté en miniature et l'utilisateur peut ainsi bénéficier d'une vue d'ensemble pour voir, par exemple, où se situe un objet ou la représentation d'un utilisateur, mais peut également interagir avec un objet dans le monde miniature et voir les résultats dans le monde de grande taille et *vice versa* (cf. figure 1.2).

[MBS97] proposent de diminuer la taille d'un monde ou d'augmenter la taille d'un utilisateur, en fonction de la position de cet utilisateur et de ses mains. Lorsque l'utilisateur relâche l'objet, le monde reprend sa taille initiale (cf. figure 1.1).

Les « poupées vaudoues[3] » [PSP99] fournissent un moyen de manipuler des objets par des représentations miniatures. La main non-dominante[4] détermine le contexte de la manipulation tandis que l'autre main permet à l'utilisateur de manipuler un objet par rapport à ce contexte. Par exemple, un utilisateur peut déplacer avec sa main

[1] « God's eye viewpoint ».
[2] « World In Miniature ».
[3] « Voodoo dolls ».
[4] Pour un droitier, la main non-dominante est la main gauche.

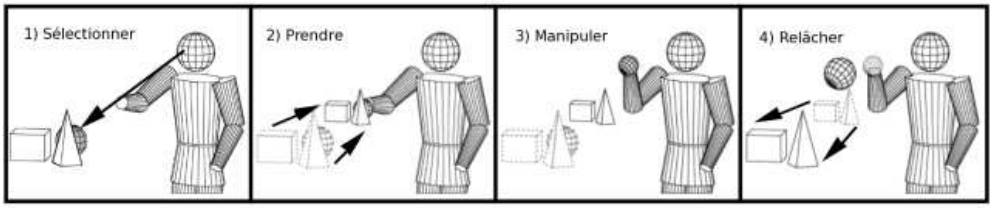


FIG. 1.1 – Principe de fonctionnement de la métaphore de redimensionnement automatique du monde appelée « Scaled-World Grab » [MBS97].

dominante le téléphone qui est sur le bureau tenu par sa main non-dominante. Lorsqu'il en a fini avec le bureau, il peut le relâcher ce qui le fait retourner à sa position (avec les modifications éventuelles apportées, ici le téléphone déplacé).

(a) (b) (c)

FIG. 1.2 – (a) Illustration d'un monde en miniature (au centre de l'image) [SCP95]. (b) Illustration du fonctionnement des « Voodoo dolls » : l'utilisateur désigne un objet grâce à une cible qui apparaît au bout de ses doigts lorsqu'il les écarte et la sélectionne en les pinçant. (c) L'utilisateur tient un objet au bout de ses doigts [PSP99].

Un positionnement extérieur par rapport à la scène virtuelle offre un point de vue général de celle-ci mais peut se révéler peu compatible avec un environnement partagé par plusieurs utilisateurs. En effet, un acteur sur le monde en miniature peut être placé à l'extérieur du monde ce qui le rend invisible pour des utilisateurs présents à l'intérieur. Dans le cas où l'utilisateur devient très grand, il peut également devenir invisible pour un utilisateur qui se situerait « entre les jambes » du géant. Par ailleurs, les actions d'un géant pourraient avoir une force décuplée (mouvements très amples) par rapport à celle d'un utilisateur de taille normale. Au lieu de faire varier la taille d'un utilisateur, c'est la taille du monde qui peut être modifiée. Dans ce cas, les ordres de variations peuvent être contradictoires entre plusieurs utilisateurs de l'environnement virtuel. D'autre part, un changement d'échelle résultant d'un utilisateur entraîne un redimensionnement de toute la scène (et éventuellement des autres utilisateurs eux-

mêmes), ce qui rompt l'échelle 1:1 ou peut entraîner des distances importantes venant gêner la navigation. Par ailleurs, il n'est pas forcément évident de se rendre compte de la variation de la taille du monde. En ce qui concerne les « Voodoo dolls », seul le résultat d'une manipulation est visible, l'outil n'est pas intégré à l'environnement. En effet, un utilisateur manipule une poupée dans ses mains et fait venir devant lui un objet, en superposition au monde. Il serait néanmoins possible de colorer d'une façon particulière un objet lorsqu'il est manipulé par une « Voodoo doll ». À cela, il faudrait aussi ajouter l'indication de quelle poupée est en train de le manipuler.

1.2.1.2 Interactions égocentriques

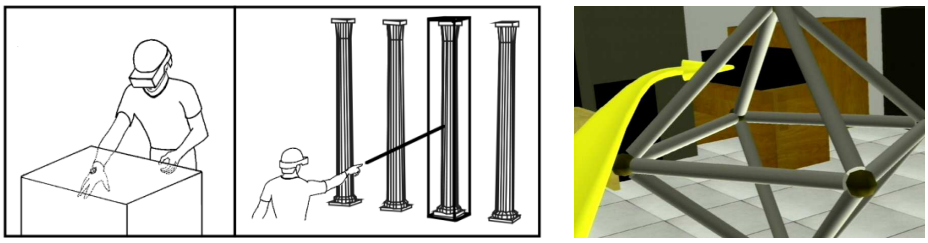
Les interactions égocentriques placent l'interacteur à l'intérieur du monde virtuel. Selon [PWBI98], les interactions peuvent être classées en deux catégories : celles de type main virtuelle et celles de type pointeur virtuel. Sous le terme « pointeur » virtuel, cet article range des techniques telles que le pointeur 3D, le rayon virtuel ou la manipulation sur des images plan. Pour éviter toute confusion, nous pensons qu'il vaut mieux parler de pointage virtuel plutôt que de pointeurs virtuels.

Main virtuelle. La méthode la plus naturelle consiste à afficher une main ressemblant à une main humaine. Sa position ou encore le mouvement de ses doigts dépendent alors de l'interface matérielle d'interaction employée par l'utilisateur comme, par exemple, un gant de données. Avec cette technique, l'utilisateur a une meilleure sensation d'immersion si la main est finement représentée parce qu'il aura l'impression que sa propre main est affichée. Un utilisateur désigne un objet en plaçant sa main au-dessus et le sélectionne, par exemple, en la refermant. Les mouvements de la main de l'utilisateur peuvent ensuite être directement imprimés sur ceux de la main virtuelle, ou être accélérés/amplifiés en fonction de l'amplitude des mouvements [Min95] pour atteindre des zones plus rapidement. Malheureusement, cette métaphore n'a qu'un champ d'action très limité. En effet, un utilisateur ne peut sélectionner un objet que s'il est à portée de sa main, sinon il doit se déplacer, ce qui peut l'amener à abandonner ses actions en cours et le ralentir dans ses travaux. Pour résoudre ce problème, les articles [PBWI96a], [PBWI96b] proposent la métaphore *Go-Go*. Le principe repose sur une extension linéaire du bras virtuel lorsque la main de l'utilisateur est proche de son corps, puis en une extension exponentielle à partir d'un certain éloignement. Par conséquent, l'utilisateur peut manipuler précisément un objet situé près de lui tandis qu'un déplacement lointain se fait par des grands mouvements. Néanmoins, cette technique limite les déplacements en profondeur de la main virtuelle à la longueur du bras virtuel étiré de l'utilisateur. Pour cette raison, [BH97] ont proposé deux nouvelles versions du Go-Go autorisant une extension infinie :

- le « *stretch Go-Go* » où l'espace entourant l'utilisateur est divisé en trois couches concentriques, la couche la plus en avant faisant allonger le bras et celle la plus en arrière le faisant raccourcir ;
- l'« *indirect Go-Go* » où l'allongement du bras est commandé par les boutons d'une souris 3D.

Cependant, un bras (réel) de l'utilisateur ne peut s'allonger, ainsi la colocalisation[5] du bras de l'utilisateur n'est plus possible.

Pointeur virtuel. En bureautique, un utilisateur utilise généralement une souris 2D pour déplacer un curseur 2D à l'écran. Dans un environnement virtuel en 3D, le curseur 2D précédent se déplace à la « surface » de l'écran et un objet n'est alors sélectionnable que s'il est projeté sur cette surface [Min95] (cf. figure 1.3). Pour pouvoir désigner des objets occultés ou lointains, il faut un curseur 3D pouvant se déplacer dans tout l'environnement 3D. Un curseur est donc un moyen abstrait de sélection. Par ailleurs, il permet de lever naturellement des contraintes : par exemple le déplacement d'un nombre élevé d'objets à la fois contrairement à une main qui sera limitée à un ou deux objets si elle respecte les contraintes du monde réel. Malheureusement, l'utilisation d'un pointeur n'est pas forcément très naturelle si elle est pratiquée comme décrite ci-avant. À la place, il serait souhaitable de pouvoir désigner un objet en le pointant avec une main [Min95] (cf. figure 1.3). Dans ce cas, l'utilisateur pourrait, par exemple, sélectionner effectivement un objet pointé en appuyant sur un bouton d'un dispositif matériel.



(a) (b) (c)

FIG. 1.3 – (a) Sélection d'un objet en approchant suffisamment un curseur 2D d'un objet projeté à la surface de l'écran. (b) Sélection d'un objet par pointage avec une main [Min95]. (c) Illustration d'un rayon coudé pointant vers un objet partiellement occulté par la structure au premier plan [OF03].

Rayon virtuel. Un rayon permet de sélectionner assez facilement un objet, même à distance si elle est raisonnable, mais il présente l'inconvénient que, une fois sélectionné, un objet reste à l'extrémité du rayon. Il devient alors très difficile de lui appliquer des rotations (à l'exception de celles autour de l'axe induit par le rayon lui-même). De plus, comme ce rayon a une longueur fixe, le positionnement en profondeur de l'objet dépend des possibilités d'allongement du bras de l'utilisateur. Néanmoins, cette contrainte trouve une solution par la méthode du « ray-casting with reeling » [BH97], c'est-à-dire par la possibilité de faire glisser l'objet sélectionné le long du rayon *via*

[5] La colocalisation du bras de l'utilisateur consiste à le superposer visuellement à sa représentation dans l'environnement virtuel. L'utilisateur ressent que son bras réel est plongé dans l'environnement virtuel.

des boutons en avant/en arrière d'un dispositif matériel. Notons que le rayon virtuel tel que présenté jusqu'à présent ne permet pas de sélectionner un objet entièrement ou partiellement occulté par un autre. Constatant que de nombreuses personnes ont tendance à plier leurs bras ou leurs poignets pour désigner un objet placé derrière un autre, [OF03] ont suggéré l'utilisation de rayons coudés (cf. figure 1.3) où l'orientation de la main de l'utilisateur désigne la courbure du rayon et l'éloignement de celle-ci par rapport au corps, la longueur du rayon.

Combinaison de métaphores. Dans les illustrations données précédemment, la facilité de sélection de ces techniques a été mise en avant ainsi que les problèmes de manipulation qu'elles posent. Par conséquent, ces techniques savent surtout se montrer pratiques pour sélectionner des objets, mêmes s'ils sont (partiellement) occultés. Parmi les métaphores d'interaction égocentriques, l'une d'elles utilise à la fois une main virtuelle et un pointage virtuel : il s'agit de la méthode HOMER[6] [BH97]. Cette technique utilise un rayon virtuel pour sélectionner un objet mais, ensuite, une main virtuelle va venir s'attacher à l'objet pour pouvoir le manipuler. Le but de cette combinaison est de pouvoir profiter de la précision de sélection offerte par un rayon virtuel et de la facilité de manipulation apportée par une main virtuelle. Dans le même temps, les difficultés de manipulation provenant de l'usage d'un rayon virtuel et les difficultés de sélection d'objets éloignés par une main virtuelle disparaissent.

Selon les variantes de cette technique, l'utilisateur peut ramener un objet distant à lui en rapprochant sa main physique de son corps — il faut alors que le facteur d'échelle appliqué soit adapté pour permettre une telle action — ou en actionnant des contrôles d'un périphérique (dans [BH97], les boutons d'une souris sont employés). La fin de la manipulation permet à la main virtuelle de reprendre sa position d'origine, c'est-à-dire au niveau de l'avatar de l'utilisateur. Cette technique regroupe les qualités des techniques de sélection par rayon virtuel et des techniques de manipulation par une main virtuelle.

Techniques sur images plan. Pierce et *al.* [PFC+97] proposent à un utilisateur de sélectionner ou manipuler des objets en interagissant avec la projection 2D d'une scène virtuelle 3D. Cette technique est à employer dans le cas où les mouvements de la tête de l'utilisateur sont suivis. Un utilisateur sélectionne un objet virtuel grâce à la projection de ses doigts sur l'objet affiché (cf. figures 1.4) :

- « *Head Crusher* » : l'utilisateur sélectionne un objet lorsqu'il est compris entre son pouce et son index ;
- « *Sticky Finger* » : l'utilisateur passe son index sur l'objet à sélectionner ;
- « *Lifting Palm* » : l'utilisateur sélectionne l'objet qui repose dans la paume de sa main ;
- « *Framing Hands* » : l'utilisateur sélectionne l'objet qui est compris entre ses deux mains.

[6] *Hand-centered Object Manipulation Extending Ray-casting.*

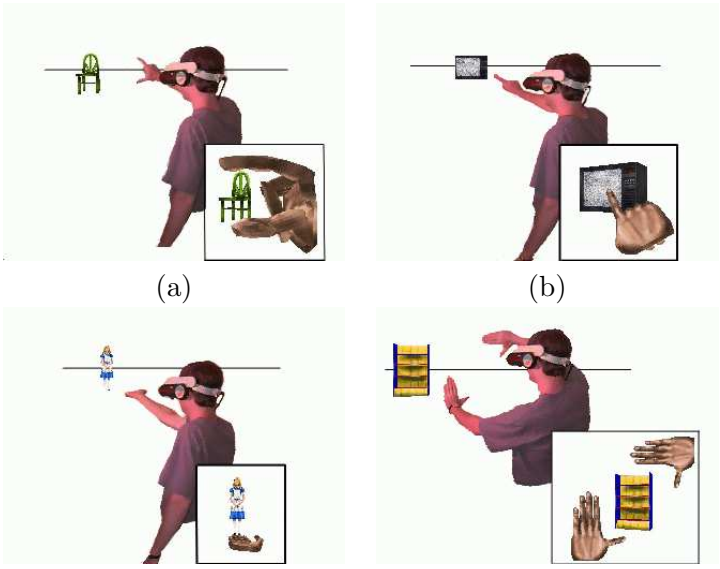


FIG. 1.4 – Illustrations des techniques de sélection sur images plan : (a) « Head Crusher », (b) « Sticky Finger », (c) « Lifting Palm » et (d) « Framing Hands ».

Usages des techniques d'interaction selon différents systèmes d'affichage. Les techniques d'interaction qui ont été présentées peuvent s'employer dans différentes conditions d'affichage. Nous proposons d'essayer d'évaluer dans quelles conditions d'affichage ces techniques peuvent être les plus pertinentes. Nous allons considérer deux conditions d'affichage : l'affichage par projection, c'est-à-dire lorsque l'image est projetée sur un grand écran placé face aux utilisateurs, et l'affichage au moyen d'un casque ou d'une paire de lunettes présentant une image différente pour chaque œil[7]. Nous faisons ici également la différence entre les images monoscopiques et stéréoscopiques.

La table 1.1 exprime notre opinion en ce qui concerne les cas où une technique d'interaction est plus adaptée qu'une autre.

Ainsi, la main virtuelle se révèle faiblement adaptée en cas d'usage d'une projection parce que l'utilisateur verra à la fois sa main réelle et sa main virtuelle. Dans certaines postures, la main réelle pourrait créer une occlusion de la main virtuelle. Quant à la stéréoscopie, elle aide à évaluer la profondeur et facilite donc l'usage de la technique. Le Go-go présente les mêmes inconvénients que la main virtuelle « classique ».

La manipulation d'un pointeur virtuel ou d'un rayon virtuel peut s'effectuer par différents périphériques d'interaction puisque le but n'est pas de mimer la réalité mais plutôt de s'en éloigner. Il est donc difficile de considérer que ces deux techniques ne seront influencées autrement que par la stéréoscopie pour l'appréciation de la profondeur, ou par des contraintes relatives à une tâche particulière.

[7]En anglais, il s'agit d'un Head-Mounted Display (HMD).

Pour HOMER, les inconvénients de la main virtuelle se retrouvent, c'est-à-dire une occlusion possible de la main virtuelle par la main réelle.

Pour une image plan, la stéréoscopie entraîne une difficulté pour la sélection d'un objet. En effet la différence entre ce que voit l'œil droit et ce que voit l'œil gauche peut engendrer une imprécision importante. De fait, plusieurs solutions possibles sont données dans [PFC+97] mais les auteurs ne parlent que de résultats obtenus en monoscopie. Par conséquent, nous ne donnons pas d'appréciation concernant l'usage de la stéréoscopie.

	Projection écran		Casque	
	Mono	Stéréo	Mono	Stéréo
Main virtuelle	--	-	+	++
Go-go	--	-	+	++
Pointeur virtuel	+	++	+	++
Rayon virtuel	+	++	+	++
HOMER	--	-	+	++
Images plan	-	?	+	?

TAB. 1.1 – Comparaison des techniques d'interaction selon différents systèmes d'affichage : projection sur un écran et affichage par un casque, par exemple à cristaux liquides, pouvant présenter une image différente à chaque œil. De plus, il est fait une différence entre un affichage mono et un affichage stéréoscopique. Avec le symbole « -- », une technique sera moins appropriée que si elle a le symbole « - », « + » ou « ++ » par ordre croissant. Dans le cas du symbole « ? », aucun résultat n'est donné.

1.2.1.3 Bilan intermédiaire

Un panorama plus détaillé des techniques d'interaction à un utilisateur est disponible dans [BKLJP05]. Les métaphores exocentriques entraînent un détachement de l'utilisateur par rapport à l'environnement virtuel. Les métaphores égocentriques permettent au contraire de mettre les utilisateurs à l'intérieur de l'environnement virtuel, ce qui peut entraîner une meilleure sensation d'immersion et faciliter l'interaction dans un environnement virtuel.

1.2.2 Perception du potentiel interactif des objets virtuels

Pour pouvoir interagir avec un objet, tout interacteur a besoin de connaître les possibilités interactives qu'offre cet objet :

- les propriétés modifiables de l'objet (position, couleur, etc.) ;
- les zones interactives : où le saisir, comment, à combien de personnes, etc. ;
- les limites et contraintes d'interaction : impossibilité de faire traverser un mur à un objet, mouvements contraints par rapport à un autre objet, etc.

Ces informations peuvent utiliser des retours haptiques, visuels ou sonores qui se rajoutent à l'environnement virtuel.

1.2.2.1 Propriétés modifiables

L'une des premières interactions qu'un objet peut offrir à un interacteur est celle de son propre déplacement ou des rotations qu'il peut subir. L'un des premiers systèmes à afficher le fait qu'un objet puisse être translaté ou orienté a été Inventor de SGI[8]. Ce kit de développement permettait de construire facilement des mondes en trois dimensions grâce à l'utilisation et l'extension d'objets fournis. Inventor affichait une sorte de curseur dont la forme indiquait les translations, rotations ou changements d'échelles possibles (cf. figure 1.5). Cette idée a été souvent exploitée dans les logiciels de CAO. Dans le domaine de la réalité virtuelle, [DLT04] en fournit un exemple. Cependant, ce type d'information est rarement affiché dans les environnements virtuels, notamment parce qu'il surcharge le monde visuel.

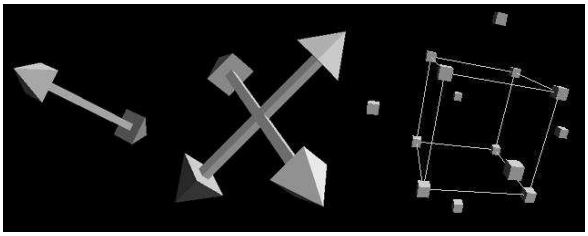


FIG. 1.5 – Trois curseurs utilisés par Inventor de SGI. Les deux premiers indiquent les translations possibles. Le dernier, le plus à droite, permet de changer l'échelle d'un objet.

Enfin, outre l'affichage d'un symbole représentant les degrés de liberté d'un objet, il est également possible de donner ces informations implicitement par l'affichage de l'outil d'interaction qui est strictement réservé à une translation ou à une rotation. Dans [CFH97], par exemple, les outils sont représentés par des icônes qui n'apparaissent que lorsqu'une des opérations qu'ils peuvent effectuer devient accessible. Un utilisateur peut alors sélectionner une icône pour employer l'outil associé.

1.2.2.2 Objets et parties interactives

Pour pouvoir interagir avec un objet, un interacteur doit savoir si l'objet est interactif et en connaître les parties interactives. Par exemple, une porte peut être poussée pour l'ouvrir et sa poignée peut être tournée. Néanmoins, une porte ne pourra être poussée que si la poignée a été actionnée. Il faut donc fournir à l'interacteur un moyen de savoir que la poignée peut être tournée et la porte poussée. Cependant, il faut aussi montrer à l'utilisateur que la porte ne peut être poussée que si la poignée a été tournée.

La représentation des parties interactives d'un objet sert à fournir à un utilisateur humain les informations nécessaires à une interaction avec un objet, mais elle peut aussi permettre d'aboutir à une perception « logicielle » rendant des agents autonomes

[8] `http://oss.sgi.com/projects/inventor/`, aujourd'hui développé par la société Mercury Computer (`http://www.mc.com/`).

capables d'interagir avec ces objets. Le concept de *smart object* [KT98], [Kal01] permet de décrire les possibilités interactives qu'offrent des objets. En outre, ces objets possèdent des comportements qui décrivent la position qu'un agent doit avoir avant de démarrer une interaction, puis lui dictent le comportement qu'il doit suivre ensuite. [BD04] font remarquer que le fait de placer toutes les informations d'interaction, ainsi que des actions, dans un objet transforme l'agent autonome en une sorte de coquille vide. Pour ne pas figer autant le comportement des agents autonomes, ils proposent le concept d'*objets synoptiques* qui offrent à un agent autonome un sommaire des interactions qu'ils fournissent, ainsi que la connaissance de leur état et des actions disponibles à partir de celui-ci. Néanmoins, ces objets prévoient rarement le cas d'interactions partagées par plusieurs agents, voire interacteurs humains.

1.2.2.3 Limites et contraintes des interactions

De la même façon que dans le monde réel, un objet du monde virtuel peut être muni de contraintes telles que le fait de ne pas pouvoir passer au travers d'un mur ou d'un autre objet. Les limites, ou contraintes, sur des mouvements peuvent donc empêcher certaines actions dans un environnement virtuel afin de reproduire des contraintes du monde réel. De plus, la limitation des mouvements due aux contraintes peut aussi rendre des actions plus précises ou plus faciles à effectuer en servant de guide géométrique : par exemple, des tremblements de l'utilisateur seront moins répercutés si la pièce qu'il manipule est contrainte à des translations le long d'un plan.

L'assemblage ou le positionnement relatif de pièces peut être délicat pour au moins trois raisons :

- soit il n'y a pas de système de détection de collisions et donc les pièces se traversent les unes les autres ;
- soit pour une raison ou une autre l'utilisateur ne peut pas placer son point de vue à sa guise et voit mal les placements des pièces qu'il manipule ;
- soit les interfaces matérielles qu'il utilise n'offrent pas un retour haptique et il devient difficile de correctement estimer l'instant et les conséquences du contact.

Une solution aidant l'utilisateur à emboîter des pièces virtuelles et à les placer précisément peut consister à limiter les degrés de liberté de la pièce qu'il manipule. Dans VLEGO [KTK+98], par exemple, le déplacement d'une pièce est toujours contraint. En effet, chaque pièce est déplacée selon une grille afin de pouvoir être emboîtée avec les autres (cf.figure 1.6). Une pièce comporte alors 4 degrés de liberté (3 en translation et 1 en rotation).

Les contraintes peuvent donc apparaître comme une aide, un guidage : il faut alors permettre à l'utilisateur de faire la part des choses entre les contraintes qui peuvent l'aider et celles qui modélisent de vraies limitations. Cependant, dans les deux cas, les retours exprimant les limitations sont de trois natures : haptique, visuelle et sonore.

Retour haptique. La restitution haptique peut transmettre une contrainte au moment d'un contact ou d'une collision. Un bras à retour d'efforts va pouvoir freiner le mouvement d'un utilisateur lorsque ce dernier entrechoquera la pièce qu'il manipule

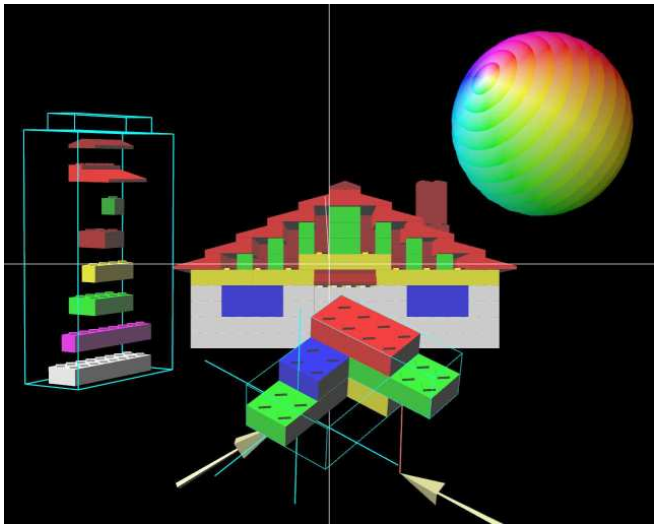


FIG. 1.6 – Illustration d'un assemblage de pièces dans VLEGO [KTK+97]. Les primitives sont placées à gauche. Les deux curseurs sont les flèches autour du grand cube blanc central. La boule colorée permet de sélectionner une couleur à appliquer à une pièce.

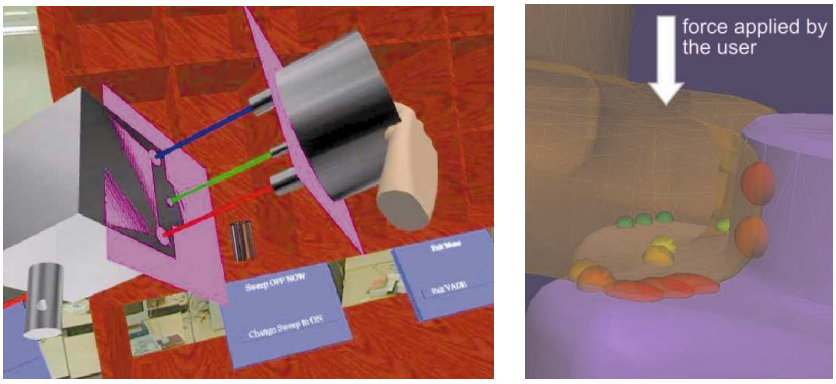
avec une autre pièce présente dans l'environnement virtuel. D'autres systèmes permettent à l'interacteur de « ressentir », restituer, un contact de son propre corps[9] avec un objet virtuel (utilisation d'un système vibrotactile [BB07]). L'haptique se divise donc en deux types de retours : le retour de force et le retour tactile. La décision d'utiliser un système à retour d'efforts ou tactile peut dépendre de contraintes pratiques. Les systèmes tactiles sont généralement plus légers et plus facilement transportables que les systèmes à retours d'efforts même s'il existe des versions plus compactes de systèmes à retour d'efforts, comme par exemple le Spidar[10] [KIKS00]. Un bras à retour d'efforts précis est cher, souvent encombrant et d'une amplitude de mouvements faible. Le choix d'un système à retour d'efforts dépend également des objectifs d'une expérience : un système tactile permet surtout d'avoir connaissance de la présence d'un contact alors qu'un système à retour de force permet de mieux évaluer la grandeur d'une force [MS93]. Le coût, la faible précision de certains systèmes, leur encombrement, ou la difficulté de leur mise en œuvre (coûts de calculs de la simulation élevés, nécessité de disposer de données physiques pour les objets simulés, etc.), font que l'usage des systèmes haptiques n'est pas toujours possible.

Retour visuel. Pour les raisons évoquées précédemment, beaucoup d'utilisateurs d'environnements virtuels profitent uniquement d'indications visuelles pour comprendre les contraintes en présence (cf. figure 1.7) :

[9] Bien entendu, il faut ici imaginer une immersion du corps de l'utilisateur dans l'environnement virtuel.

[10] *SPace Interface Device for Artificial Reality.*

– d'une trajectoire à respecter ;
– d'une zone dans laquelle les déplacements d'une pièce sont autorisés ;
– de butées.



(a) (b)

FIG. 1.7 – (a) La contrainte sur les mouvements d'un objet est matérialisée par des lignes droites entre les deux pièces qui représentent la trajectoire [JJW+99]. (b) Les forces appliquées sont représentées par des sphères colorées qui s'aplatissent [SLMA06].

Par ailleurs, il est possible de combiner une interface haptique passive à un retour visuel adapté afin de simuler des propriétés haptiques ; ce type de retour est qualifié de *pseudo-haptique* [LCK+00]. Ainsi, sur un périphérique isotonique[11], l'utilisateur doit effectuer de grands mouvements pour faire un peu bouger un objet qui est freiné par une friction dans l'environnement virtuel : la difficulté pour le déplacement amène l'utilisateur à imaginer une friction.

Retour sonore. Le retour sonore peut être utilisé pour faire ressentir à un utilisateur des contraintes. Un premier moyen de représentation d'une contrainte peut être apporté par la simple volonté d'imiter la réalité. En fonction du réalisme des sons, les utilisateurs de l'environnement virtuel pourront avoir une interprétation, d'où une réaction, naturelle. Par exemple, l'impossibilité de faire pénétrer un objet dans un autre (ex : un ballon dans un mur) peut être signalée par un son normalement entendu dans la réalité lorsque ces deux objets s'entrechoquent. La représentation d'une contrainte peut aussi être obtenue par un retour sonore plus abstrait. Par exemple, un son peut être produit lorsqu'un objet manipulé s'éloigne d'une trajectoire ou d'une position souhaitée. La hauteur de ce son peut être modulée en fonction de la distance entre l'objet et cette position. Un autre exemple est l'utilisation de bipeurs. Lorsque l'objet manipulé approche d'une position non-souhaitée, un son est produit pour signaler à l'utilisateur sa mauvaise opération. Au contraire, un son pourrait aussi être émis pour signaler une opération correcte. Par ailleurs, l'aide apportée par le retour sonore peut prendre

[11] La position d'un objet manipulé par un périphérique isotonique est fonction des déplacements physiques de ce périphérique. Une souris 2D est un périphérique isotonique.

plus d'importance avec des utilisateurs ayant des difficultés visuelles (pour aider au déplacement ou valider des actions effectuées).

Un retour sonore peut améliorer les performances dans la réalisation d'une tâche d'assemblage de pièces comme dans [MS93] où l'expérimentation qui consiste à enfiler une cheville dans un trou, opération nommée « Peg-in-Hole », est réalisée plus rapidement avec un retour sonore. Ce retour intervient pour signaler un choc (son ponctuel) ou la grandeur d'une force (variation des fréquences basses du son).

Retour multimodal. Enfin, une tendance très forte est actuellement de combiner de différentes façons des retours visuels, sonores et haptiques comme dans [ZSF05] où les utilisateurs effectuent l'expérience « Peg-in-Hole » plus rapidement lorsqu'ils ont un retour visuel et sonore combiné que lorsqu'ils n'ont qu'une seule modalité de retour. L'article [HRH06] conclut que la multiplication des modalités simultanées améliore les performances d'un utilisateur dans un environnement virtuel (tant que le nombre de ces modalités reste peu élevé pour ne pas surcharger le système cognitif d'un individu). Cette expérience emploie des retours haptiques, visuels et sonores. Dans [SBLG+07], de multiples retours (cf. figure 1.8 pour les retours visuels) sont utilisés lorsque, par exemple, une roue crantée impacte une barre : un jet de particules ayant le point de collision comme origine apparaît, un son bref représentant le choc est émis, le bras à retours d'efforts bloque le mouvement de l'utilisateur. Les retours visuels et sonores permettent ici de renforcer la sensation de contact donnée à l'utilisateur.

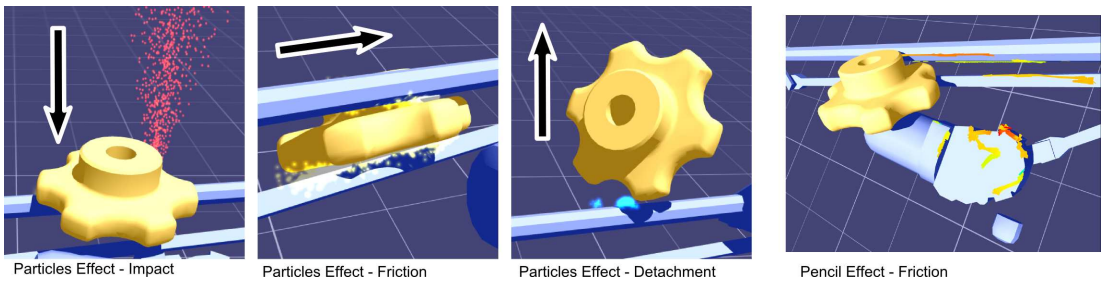


FIG. 1.8 – Illustration de collisions [SBLG+07].

1.2.2.4 Bilan intermédiaire

Un utilisateur doit être informé du potentiel interactif des objets pour lui permettre de comprendre comment interagir avec un objet et même améliorer ses performances en réalisant des opérations plus rapidement avec cet objet, que cet objet soit un outil ou non. Il faut donc permettre à l'utilisateur de déterminer si le déplacement ou le changement d'orientation d'un objet est possible. Ensuite, l'utilisateur devra être doté de moyens (d'outils) permettant d'exprimer des nouvelles positions ou orientations afin de modifier ces propriétés. Il en est de même pour toutes les propriétés de l'objet (comme la couleur par exemple) : l'utilisateur doit être averti du potentiel interactif d'un objet

et de la façon de l'utiliser, c'est-à-dire avec quel outil interagir, ou bien quelles sont les opérations applicables. Cependant, afin d'éviter à l'utilisateur des mouvements ou des opérations inutiles, afin même de ne pas lui permettre certaines actions pour mieux mimer des contraintes physiques du monde réel, certaines opérations doivent rester impossibles ou limitées. Ce guidage peut être obtenu par des retours haptiques parce qu'ils permettent d'empêcher des mouvements de l'utilisateur. Par exemple, un utilisateur placera plus aisément une pièce virtuelle le long d'un conduit virtuel grâce à ces contraintes haptiques. Des retours sonores peuvent également être produits pour guider un utilisateur. Des sons réalistes tendent à donner la possibilité aux utilisateurs d'avoir des réactions naturelles par rapport à ceux qu'ils entendent. Par exemple, le bruit de deux objets qui s'entrechoquent tend à indiquer qu'ils ne peuvent s'interpénétrer, donc que le déplacement souhaité est impossible. À des sons réalistes peuvent être ajoutés des sons abstraits qui, en fonction du contexte, peuvent indiquer le succès ou l'échec d'une action. Des retours peuvent également aider un utilisateur à effectuer une opération plus rapidement. Enfin, certains retours pourraient, *a priori*, être envoyés aux utilisateurs partageant une interaction, notamment les retours visuels ou sonores s'ils ne sont pas trop nombreux et s'ils ne font pas confondre à l'utilisateur le résultat de ses actions avec celui d'autres personnes.

1.3 Utilisation des deux mains : étape vers le collaboratif

Le collaboratif autorise plusieurs utilisateurs, ou outils d'interaction, à agir simultanément sur un même objet virtuel. L'usage de deux mains pour un même objet virtuel fait donc apparaître une situation de collaboration. Nous verrons ici des métaphores d'interaction où il n'y a qu'un seul outil divisé en deux parties par l'utilisation de deux mains. Il s'agira donc ici de faire un pas en direction d'une collaboration plus complète.

Dès 1986, Buxton et Myers [BM86] proposaient une évaluation de l'apport de l'utilisation de deux mains pour des opérations de navigations/sélection dans un espace 2D. Par rapport aux environnements virtuels en 3D, l'expérience était plutôt rudimentaire mais montrait déjà que la possibilité de diviser une tâche en sous-tâches effectuées par deux mains en même temps, et avec une continuité entre ces sous-tâches, permettait d'aller plus vite que dans le cas où une seule main aurait agi.

Cependant, l'idée, généralement répandue, selon laquelle le travail bimanuel où les mains travaillent en parallèle ne sert qu'à faire gagner du temps n'est pas une façon correcte d'aborder les interfaces bimanuelles. D'abord, deux mains ne travaillent pas forcément en parallèle, donc l'usage de deux mains ne permet pas d'aller systématiquement deux fois plus vite. Ensuite, l'utilisation de deux mains peut faire gagner plus que du temps : explorer l'environnement, ou aider l'utilisateur lui-même à structurer sa pensée en créant une sorte de boîte englobante à partir de sa deuxième main par exemple — l'utilisateur utilise alors l'une de ses main comme un référentiel pour son autre main [HPPK98]. Par conséquent, il existe deux familles de métaphores d'interaction bimanuelles, celles où les interactions bimanuelles sont asymétriques et celles où elles sont symétriques.

1.3.1 Interactions bimanuelles asymétriques

Les travaux menés dans la famille d'interactions bimanuelles asymétriques (dites aussi *coopératives* [BKLJP05]) peuvent être divisés en deux parties : dessin/modélisation/assemblage de pièces, puis déplacement/manipulation/auscultation d'objets.

1.3.1.1 Dessin/modélisation/assemblage de pièces

Les travaux décrits dans cette partie permettent de s'affranchir de certaines difficultés imposées par la 2D lors de la modélisation d'objets en 3D. En effet, cette situation impose une traduction fastidieuse de toute opération 3D en plusieurs sous-opérations 2D. Pourtant, la plupart des logiciels de CAO restent actuellement limités à des interfaces en 2D.

3-Draw [SRS91] est un système permettant de dessiner des esquisses en 3D à partir de courbes 3D puis de pouvoir les déformer (étirer, tordre, couper, etc.). L'hypothèse utilisée par cette application est que toute personne sait situer dans l'espace l'une de ses mains par rapport à l'autre et que cette information peut être utilisée par l'utilisateur afin de lui fournir une façon plus naturelle de manipuler des objets virtuels. L'utilisateur tient dans une main une tablette et dans l'autre un stylet. La tablette sert de référentiel sur lequel le stylet permet de dessiner. Par ailleurs, de nouvelles orientations de l'objet affiché sont obtenues par des mouvements de la tablette.

De la même façon, THRED[12] [SG94] est un système permettant de modifier des terrains, en 3D. Une surface se construit en subdivisant une surface par tesselation[13], ou en ajoutant des nouveaux rectangles, ensuite tesselables, à celui initialement placé. La main droite (ici, la main dominante) permet la sélection de points de contrôle, le déplacement de ces points pour modifier la géométrie de l'objet courant, et le lancement d'une nouvelle tesselation à un niveau de détails plus ou moins élevé. La main gauche (main non-dominante) permet de sélectionner l'utilisation d'une contrainte, par exemple le déplacement d'un sommet peut être limité au déplacement le long d'une ligne pour faciliter son placement au lieu de rester sous 3 DDL. Elle permet également de sélectionner un niveau de tesselation précédemment obtenu, de changer la position et l'orientation de l'ensemble de la scène, et enfin de choisir un modificateur de forme via un menu circulaire (forme conique ou pyramidale).

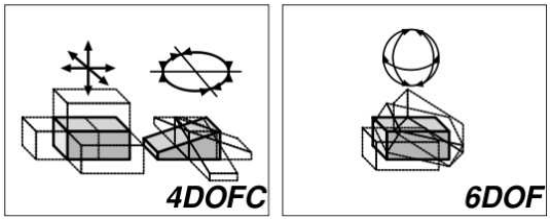
Dans VLEGO[14] [KTK+98], l'utilisateur modélise des objets en assemblant des primitives en forme de pièces de Lego^{TM}. Une illustration est donnée sur la figure 1.6. Lorsqu'un utilisateur n'emploie qu'une main, il déplace une pièce virtuelle selon 4 DDL (cf. figure 1.9 (a)). Les trois premiers degrés de liberté correspondent à des déplacements en translation sur une grille virtuelle s'étendant à l'ensemble de la scène virtuelle. Le dernier degré de liberté est une rotation autour de l'azimut par des angles pré-déterminés. Lorsqu'un utilisateur emploie ses deux mains, deux modes de placement de pièces virtuelles sont possibles. En mode dit « séparatif », les deux pièces manipulées, chacune

[12] *Two Handed Refining EDitor*.

[13] La tesselation d'une surface consiste à ajouter des sommets supplémentaires sur celle-ci sans en changer sa topologie.

[14] VirtuaL Environment for Generating Objects

par une seule main, sont distantes et sont manipulables selon 4 DDL comme pour une manipulation à une seule main. En mode dit « coopératif », les pièces virtuelles sont proches. Dans ce mode, il est considéré que l'on cherche à emboîter ces deux pièces et qu'utiliser plus de 4 DDL facilite la tâche. Pour ce faire, l'objet virtuel associé à la main non-dominante est considéré comme étant la base et dispose de 6 DDL. L'objet virtuel manipulé par la main dominante ne dispose que de 4 DDL avec une base de référence qui n'est plus la grille comme précédemment mais la pièce tenue par la main non-dominante. La fin de manipulation d'une pièce virtuelle la contraint à s'aligner avec la grille virtuelle de l'ensemble de la scène.



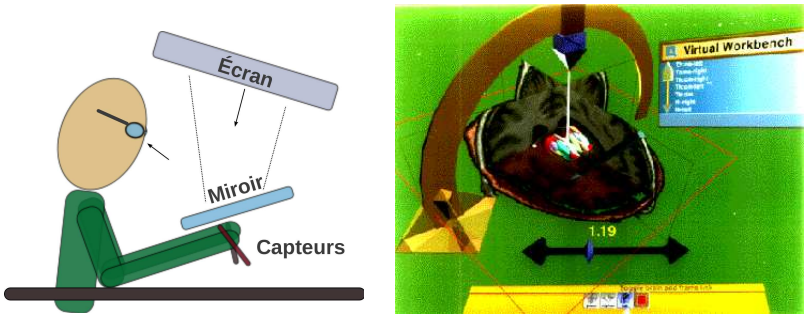
(a) (b)

FIG. 1.9 – Illustration des degrés de liberté disponibles dans VLEGO [KTK+98] pour une pièce. (a) Les rotations sont limitées à un seul axe. (b) Une pièce dont les mouvements ne sont pas limitée.

1.3.1.2 Déplacement/manipulation/auscultation d'objets

Le recours à des techniques permettant d'observer des objets sous « toutes les coutures » est particulièrement présent dans le domaine de la chirurgie. Poston et Serra [PS96] proposent à la fois du matériel et des techniques d'interaction permettant d'appliquer, entre autres, des rotations et des coupes à un cerveau humain. Une interface pouvant être tenue par un utilisateur pour lui donner l'impression de tenir, réellement, l'objet virtuel s'appelle une *interface tangible*. Ici, le matériel, nommé « *Brain Bench* », comprend un miroir (opaque) projetant l'image d'un écran en dessous duquel l'utilisateur place ses mains (cf. figure 1.10 (a)). L'utilisateur voit par réflexion sur le miroir une représentation synthétique des outils qu'il tient et a ainsi la sensation que les outils sont dans la scène et surtout devant lui, sous ses yeux. Les interactions se déroulent de cette façon pour examiner un cerveau humain : avec sa main gauche (main non-dominante), l'utilisateur peut prendre, déplacer et tourner un cerveau pendant que sa main droite (main dominante) tient un stylet qui déplace un plan de coupe (cf. figure 1.10 (b)).

Hinckley et *al.* [HPPK98] proposent également de pouvoir déplacer un cerveau virtuel pour lui appliquer des coupes mais, contrairement à Poston et Serra [PS96] qui utilisent des stylets, ils ont recours à une tête de poupée et une tranche de plastique. La tête permet à l'utilisateur de s'imaginer en train de tenir le cerveau et d'indiquer dessus la coupe à appliquer avec le morceau de plastique (cf. figure 1.11).



(a) (b)

FIG. 1.10 – (a) Organisation schématique du « Brain Bench » [PS96]. Un utilisateur voit sur un miroir (opaque) les images stéréoscopiques affichées par un moniteur. En dessous du miroir, l'utilisateur tient des capteurs à 6 DDL montés sur des stylets. Ces stylets (ou une autre représentation) sont reproduits, virtuellement, en 3D sur l'image de l'écran. (b) Exemple d'une image projetée [PS96].

Par l'utilisation d'interfaces tangibles, l'utilisateur dispose d'un moyen très naturel d'interagir avec un objet parce qu'il peut avoir une meilleure impression de le tenir dans ses mains, d'où une meilleure précision et une meilleure sensation d'immersion. Ces interfaces permettent aussi d'offrir de meilleures performances qu'une IHM[15] graphique 2D où il y auraient de multiples contrôles pour des translations/rotations selon les différents axes du repère 3D utilisé.

1.3.2 Interactions bimanuelles symétriques

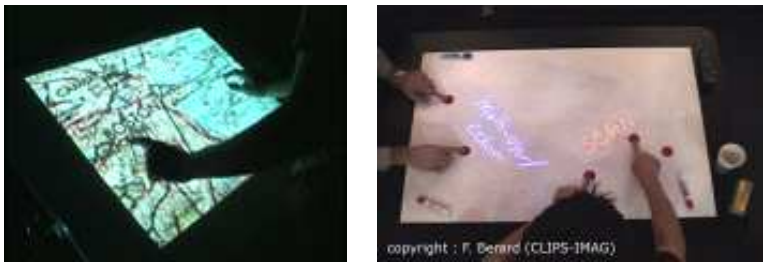
Avec ce type de techniques d'interaction, l'utilisateur effectue des mouvements/tâches similaires avec ses deux mains en parallèle.

Dès 1985, Krueger [KGH85, Kru93] était pionnier et utilisait un système de suivi vidéo des mouvements et des positions de personnes grâce à VIDEOPLACE. Il imaginait alors des utilisateurs éloignant deux doigts posés sur un écran pour effectuer un zoom dans la zone rectangulaire alors définie.

Aujourd'hui, les écrans multi-tactiles sont en train de se démocratiser. Ainsi, nous pouvons citer les produits de la société Perceptive Pixel[16]. Dans le même genre, le projet DIGITABLE[17], financé par l'ANR pour la période 2006–2008, utilise un projecteur vidéo dirigé vers un support multi-tactile. Un exemple de l'interface de Perceptive Pixel en action et un autre pour DIGITABLE sont présents sur la figure 1.12. Pour le grand public, on peut également parler de la table Surface™ de Microsoft [Sur]. Mais pour ce même grand public, c'est surtout l'usage de terminaux mobiles tels que l'iPhone™ ou de ceux basés sur Google Android™ qui répand ce type d'interaction.

[15] Interface Homme-Machine.
[16] `http://www.perceptivepixel.com/`
[17] `http://www.digitable.fr/`

FIG. 1.11 – Les périphériques utilisés par Hinckley et *al.* [HPPK98] pour effectuer une coupe sur l'imagerie d'un cerveau. La tête de poupée et la plaque en plastique, qui est l'outil de coupe, sont suivies en positions et orientations.



(a) (b)

FIG. 1.12 – (a) L'interface de Perceptive Pixel où un utilisateur consulte une carte en la déplaçant avec deux doigts. (b) Table DIGITABLE où les utilisateurs peuvent agrandir un texte grâce aux deux jetons dont ils disposent.

Cutler et *al.* [CFH97] décrivent une manipulation à deux mains sur le « Responsive Workbench », qui est une table où est projetée une image stéréoscopique avec laquelle il est possible d'interagir (cf. figure 1.13). Cette table n'est pas tactile. Les utilisateurs tiennent des capteurs 6 DDL, ou emploient des gants de données pour pincer. Cette table offre plusieurs techniques bimanuelles symétriques telles que « grab-and-carry », « grab-and-twirl ». Elle présente également une méthode bimanuelle asymétrique nommée « trackball ». Nous reviendrons sur ces techniques au chapitre 4.

Ces interfaces permettent aux utilisateurs d'accéder à des mouvements simples, faciles à associer à des actions simples : souvent, il s'agit d'appliquer des rotations, des agrandissements ou encore des déplacements à des objets virtuels.

1.3.3 Bilan intermédiaire

Les motivations pour le développement de techniques d'interaction bimanuelles sont multiples. Il s'agit d'abord de rendre les interactions plus naturelles en partant du constat que dans le monde réel les personnes cherchent généralement à utiliser leurs

(a) (b) (c)

FIG. 1.13 – (a) Présentation de l'ensemble du « Responsive Workbench » [CFH97] avec sa table transparente. En dessous, un miroir réfléchit l'image stéréoscopique d'un projecteur. (b) Un utilisateur tient des capteurs à 6 DDL dans ses deux mains. Les mouvements de sa tête sont également suivis pour changer son point de vue à l'intérieur de la scène virtuelle. (c) Gros plan sur la partie de la table qui permet de sélectionner un outil d'interaction.

deux mains. Ces manipulations plus naturelles permettent alors à l'utilisateur de reproduire des mouvements dont il a l'habitude et d'améliorer ainsi sa dextérité au sein d'un environnement virtuel. De plus, l'utilisation des deux mains améliore généralement les performances pour la réalisation d'une tâche réelle ; il s'agit alors de profiter de ce gain dans les environnements virtuels également.

Les techniques bimanuelles symétriques se limitent à des actions plutôt simples telles que le glissement d'un plan, un changement d'échelle, un déplacement ou une rotation d'objets virtuels.

Quant aux interfaces tangibles, elles sont un prolongement vers la partie matérielle des objectifs recherchés dans les interactions bimanuelles : des interactions plus faciles à appréhender et plus précises. Elles permettent en outre d'éviter des interfaces graphiques complexes où il y aurait de multiples contrôles pour les translations/rotations selon les différents axes du repère 3D de l'environnement virtuel.

1.4 La coopération

Cette section décrit les fonctions de la coopération, les méthodes d'interactions coopératives, et la prise de conscience de soi et des autres. Il y a un écart sensible entre les besoins exprimés par les utilisateurs et les solutions existantes qui seront présentées. En effet, des industriels pourraient souhaiter que leurs employés puissent reproduire dans des environnements virtuels des actions du monde réel comme des tâches coopératives d'assemblages complexes, hélas la littérature se limite souvent à proposer des solutions pour seulement déplacer et orienter des objets alors que la coopération permettrait naturellement des interactions plus avancées. L'article [BGW06] montre comment un ingénieur, un designer et un responsable marketing pourraient, au même moment et de façon colocalisée, profiter des compétences de chacun : le designer peut bouger le point

de vue à l'intérieur d'une voiture pour parler de l'allure de l'habitacle pendant que l'ingénieur peut, lui aussi, manipuler des parties de l'habitacle et mieux comprendre leur réalisation, enfin la personne chargée du marketing peut enregistrer des séquences vidéos exploitables pour des supports commerciaux. Cet usage de la coopération, dans une phase de prototypage, permet de gagner du temps et réduire les coûts de conception. L'intérêt d'industriels pour les interactions coopératives se retrouve ainsi naturellement dans le projet ANR Part@ge.

1.4.1 Fonctions

Dans le projet ANR Part@ge, la coopération[18] est considérée comme « la possibilité de contribuer à une tâche globale grâce à des actions complémentaires de plusieurs utilisateurs. Ces actions peuvent se faire de différentes façons, en alternance, ou en simultané, sur un même objet ou sur des objets différents ».

La coopération amène plusieurs utilisateurs à pouvoir travailler ensemble pour réaliser une tâche grâce à des mécanismes permettant d'interagir alternativement ou simultanément. D'une façon générale, il est difficile de montrer à un utilisateur ce qu'un partenaire est en train de faire ou a fait. Un utilisateur doit avoir conscience[19] de l'autre : sa présence, ses actions passées, courantes, voire futures s'il est possible de les inférer. Lorsqu'une interaction est le résultat d'une combinaison d'autres interactions, comme c'est le cas pour la simultanéité, il apparaît un nouveau problème : celui de faire comprendre à un utilisateur que le résultat d'une de ses actions est fonction de sa combinaison avec celles d'autres utilisateurs.

1.4.2 Interactions coopératives

Les interactions coopératives entre interacteurs humains sont de deux types : soit les personnes sont dans un même endroit, disons une même salle, soit dans des lieux géographiquement distants. La coopération locale (*colocative*) entraîne la difficulté de présenter des retours visuels et sonores à un ensemble d'individus, alors qu'ils ne sont pas forcément en train d'effectuer les mêmes actions, sans les perturber. La coopération distante entraîne des latences dans les communications réseau qui peuvent devenir gênantes lorsque des utilisateurs doivent se coordonner précisément pour interagir. Les plates-formes logicielles autorisant des interacteurs répartis géographiquement doivent disposer de mécanismes de synchronisation forte.

[18] Certains auteurs distinguent « coopération » et « collaboration » en considérant que l'un désigne des actions simultanées et l'autre des actions simplement coordonnées à la résolution d'une tâche. Malheureusement, dans ces mêmes publications, les définitions d'un mot et de l'autre s'échangent au gré des auteurs. Par conséquent, nous considérerons les deux mots comme synonymes en explicitant les situations.

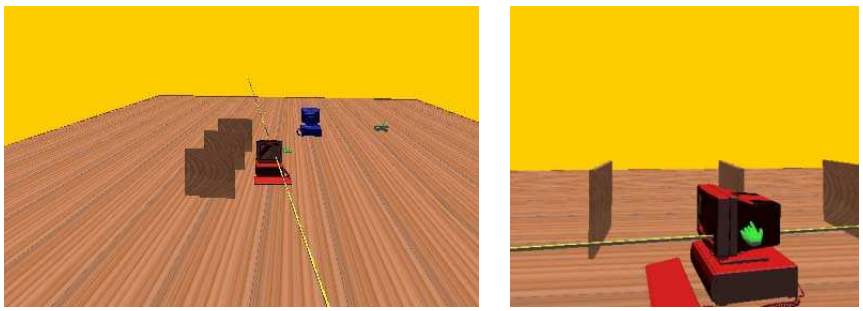
[19] En anglais, « awareness ».

1.4.2.1 La séparation des degrés de liberté

Pour réaliser des interactions coopératives, il faut combiner des commandes provenant de plusieurs interacteurs. Malheureusement, des variations dans l'ordre des manipulations géométriques (des rotations, par exemple) à appliquer peuvent conduire à des résultats finaux différents. Une telle situation peut arriver fréquemment si les interacteurs sont situés sur des lieux géographiques différents, à cause d'une latence. En séparant les degrés de liberté selon les utilisateurs, il n'est plus possible d'avoir des actions qui arrivent dans le désordre s'il est supposé que le réseau sous-jacent conserve l'ordre des informations. La séparation des degrés de liberté peut avoir lieu par deux types de techniques coopératives [PBF02] : homogènes ou hétérogènes.

Techniques coopératives homogènes. Les utilisateurs utilisent chacun une même métaphore mono-utilisateur. Dans [PBF02], les deux utilisateurs peuvent individuellement utiliser une main virtuelle ou un rayon laser virtuel.

Techniques coopératives hétérogènes. Les utilisateurs utilisent des métaphores mono-utilisateur différentes. Dans [PBF02], un utilisateur emploie une main virtuelle tandis que l'autre emploie un rayon laser virtuel. Une illustration est donnée par la figure 1.14.



(a) (b)

FIG. 1.14 – (a) Un utilisateur applique des translations à un objet, y compris le long du rayon. (b) La main virtuelle permet de faire glisser l'objet le long du rayon et de lui appliquer des rotations [PBF02].

Résultats. Les observations effectuées dans [PBF02] sont données dans le tableau 1.2. La séparation des degrés de liberté présentée dans cet article permet l'utilisation d'outils classiques (un rayon et une main virtuelle) pour coopérer. Chaque utilisateur dispose d'un point de vue sur la scène qui peut être différent. Ces méthodes s'avèrent très adaptées pour faire des ajustements et des manipulations difficiles (par exemple des manœuvres dans des espaces étroits) où elles conduisent à la réalisation plus rapide de tâches.

Tech. A	DoF de A	Tech. B	DoF de B	Commentaires
Main	Position	Main	Rotation	Utile pour des tâches d'encastrement ou des petits ajustements. Approprié lorsqu'un utilisateur ne peut pas voir des parties de l'objet virtuel manipulé.
Main	X, Y	Main	Z	Facilite le positionnement précis.
Rayon	Position	Rayon	Rotation	Utile pour les rotations qui sont difficiles avec un rayon.
Rayon	Position	Rayon	Rotation Glissement	Utile pour des placements distants et des rotations.
Main	Rotation	Rayon	Position	Utile pour les rotations qui sont difficiles avec un rayon.
Main	Rotation Glissement	Rayon	Position	Utile pour les placements distants et les rotations.

TAB. 1.2 – Observations de techniques coopératives dans [PBF02] où une main virtuelle peut être combinée avec une autre main virtuelle ou un rayon virtuel. Un rayon peut être également combiné avec un autre rayon virtuel. Chaque technique peut voir ses degrés de liberté (DoF) limités. Par exemple, une technique peut être limitée au glissement d'un objet virtuel le long d'un autre rayon virtuel.

1.4.2.2 Accès concurrent à un même degré de liberté

Nous allons présenter ici le cas où tous les degrés de liberté disponibles sont donnés à chaque interacteur suivant le classement déjà employé avec les techniques d'interaction égocentriques.

Main virtuelle. Une première méthode permettant à la fois d'effectuer des interactions collaboratives tout en conservant l'usage de la métaphore de la main virtuelle traditionnelle pourrait être d'afficher une main virtuelle dont le contrôle résulte des actions des interacteurs. Elle agirait sur le centre d'inertie de l'objet virtuel manipulé. La littérature ne présente toutefois pas cette technique, probablement parce que l'affichage d'une seule main ne donne que le résultat des actions et n'indique pas les actions en cours : il manque des retours d'informations aux interacteurs. Les auteurs s'emploient à montrer les mains virtuelles agissant sur l'objet virtuel.

L'article [NM97] décrit une méthode où plusieurs utilisateurs peuvent interagir sur un même objet au travers de bras à retour d'efforts. Chaque utilisateur actionne son propre bras à retour d'efforts et bouge ainsi la main virtuelle qui correspond à un bras. Le mouvement de l'objet est alors le résultat de l'équilibre des forces appliquées par les utilisateurs. Pour éviter des à-coups brusques suite à des délais par exemple, un système de ressort[20] est utilisé entre la main de l'utilisateur et le point d'application

[20] Ce système est un exemple de *couplage virtuel*.

sur l'objet. La figure 1.15 montre l'application de ces mains sur un objet dont l'axe de rotation, au point C, est perpendiculaire au support et évolue avec les déplacements de l'objet virtuel. Cette technique entraîne des incohérences entre la position des mains virtuelles et celle des mains réelles mais, d'après ses auteurs, si l'écart des positions est faible, il ne devrait pas gêner les utilisateurs parce que le retour visuel influence plus l'utilisateur que le retour haptique.

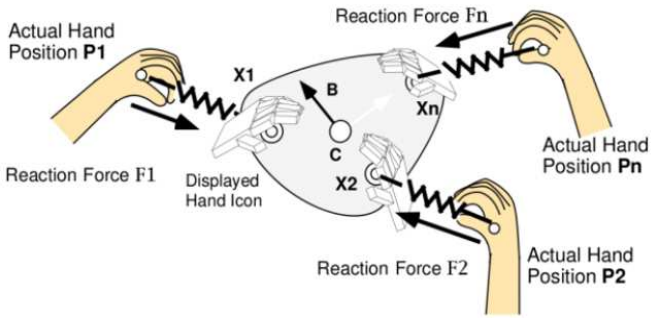
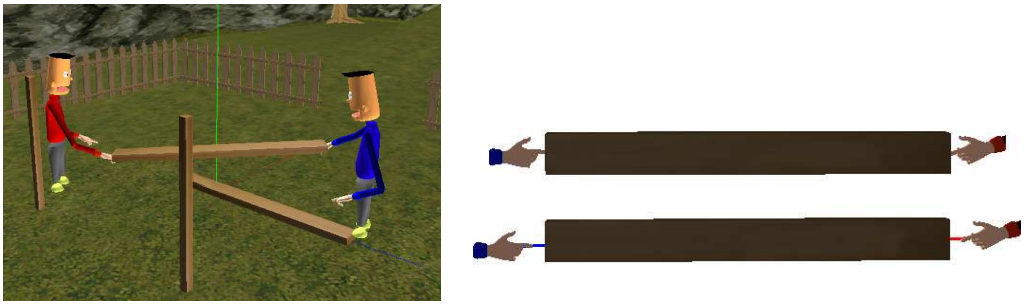


FIG. 1.15 – Application d'une force sur un objet virtuel au travers de ressorts virtuels [NM97].

Il est cependant important de noter que l'utilisation d'une main virtuelle pour le coopératif demande l'utilisation d'un retour haptique ou le recours à une métaphore. En effet, si un deuxième interacteur agit sur un objet, la main virtuelle du premier utilisateur qui a été posée dessus devra se déplacer pour suivre l'objet. Or, autant le déplacement de la main peut être ressenti par un retour d'effort, autant un simple retour visuel pourrait être difficile à utiliser pour ressentir correctement les actions de l'autre interacteur et agir rapidement en conséquence. De plus, comment un utilisateur qui se serait absenté pourrait-il comprendre que sa main virtuelle a bougé pendant qu'il n'était pas là ? Est-ce que la main doit être déformée, le bras allongé, etc. ? Dans le cas de la construction d'un belvédère virtuel [GMMG08], la saisie de chaque poutre virtuelle peut se faire par une main virtuelle d'un interacteur, l'objet est donc, dans ce cas, manipulé par deux mains virtuelles. Pour indiquer un éloignement entre la poutre et la main virtuelle, parce que les positions ne seraient pas cohérentes, des lignes rouges apparaissent (cf. figure 1.16). Si l'éloignement est trop important, la poutre devient figée jusqu'à ce que les positions des mains redeviennent vraisemblables.

Il semble que les techniques de manipulations collaboratives s'appuyant sur des mains virtuelles se limitent à l'usage d'une seule main par personne. À l'exemple donné de la construction d'un belvédère en manipulant des poutres virtuelles [GMMG08] — l'idée d'une telle construction provient de Roberts et *al.* [RWOS03a] —, nous ajoutons la manipulation d'un pare-brise virtuel par Salzmann et *al.* [SJF09] qui est aussi limitée à une main par personne.



(a) (b)

FIG. 1.16 – (a) La co-manipulation d'une poutre pour construire un belvédère virtuel [GMMG08]. Il n'y a qu'une main par interacteur. (b) Illustration des lignes rouges apparaissant entre la poutre et deux mains virtuelles.

Curseur 3D. Du fait de son attachement à représenter la réalité, la main virtuelle présente des difficultés pour représenter les effets d'actions collaboratives. Ces problèmes peuvent être en partie résolus par l'utilisation de pointeurs virtuels, au niveau d'abstraction plus élevé, comme les curseurs 3D que nous allons voir à présent.

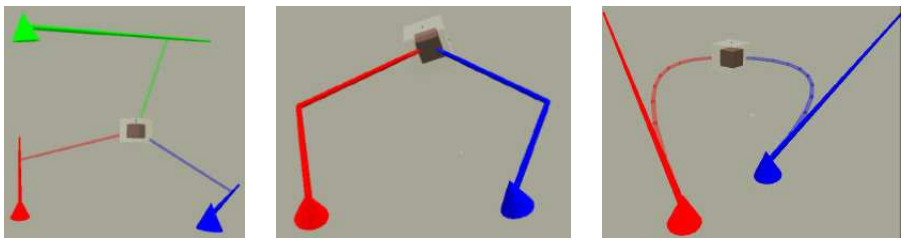
D'une façon générale, un curseur 3D est un objet 3D déplaçable dans l'espace et qui permet de sélectionner des objets (par proximité, par contact, etc.). L'article [DLT06] présente le *SkeweR* où le déplacement d'objets virtuels s'effectue par les positions de curseurs 3D. Un utilisateur approche son curseur près d'un point de contrôle pour manipuler l'objet. S'il n'y a qu'un utilisateur, seuls des mouvements de rotations autour du centre géométrique de l'objet peuvent lui être imprimés. À partir de deux utilisateurs, l'objet se manipule tel une brochette sur laquelle les utilisateurs appliquent des translations ou des rotations. Cela a l'avantage de ne pas limiter la préhension d'un objet uniquement en son centre géométrique, mais elle rend inaccessible l'emploi d'un degré de liberté induit par le « SkeweR » lui-même : la rotation autour de l'axe formé par les deux extrémités. Cette limitation disparaît avec trois utilisateurs ayant des points de contrôle formant un plan. Au-delà des trois interactions, les mouvements d'un « SkeweR » deviennent sur-contraints et nécessitent un arbitrage.

Rayon virtuel. Les rayons classiques explicités précédemment ne permettent pas d'interactions simultanées sur des objets même s'il est facile d'imaginer plusieurs rayons reliés à un même objet. En effet, si un interacteur bouge son rayon, l'objet bouge également et, soit les autres rayons vont le suivre mais ils ne vont alors pas rester cohérents avec la position imprimée par leur utilisateur, sauf s'il est possible de bouger ces rayons avec un retour d'effort, soit les autres rayons ne vont pas pouvoir suivre l'objet car ils vont rester dans la position imprimée par leur utilisateur. En l'absence de retour d'efforts, on place généralement l'objet à une position qui est le barycentre des positions proposées par les rayons et il va donc falloir montrer sur chaque rayon les contraintes résultantes de l'interaction coopérative. Une façon de traiter ce problème

est de courber les rayons en fonction des forces appliquées sur ceux-ci. L'article [DF02] propose trois types de rayons (cf. figure 1.17) :

- l'élastique qui, si un objet ne suit pas une trajectoire imposée par l'interacteur à cause d'une contrainte, force un élastique à s'étirer entre l'interacteur et l'objet ;
- le rayon coudé qui se plie vers l'objet manipulé ;
- le rayon déformable qui se dédouble avec une partie qui reste raide (celle qui a permis la sélection) et une autre qui se plie pour conserver un contact avec l'objet dont les mouvements sont contraints par la coopération ; la courbure montre l'écart entre la position que l'objet devrait avoir et celle qu'il a réellement.

Nous noterons que, parallèlement, [RHWF06] ont eux aussi proposé une métaphore basée sur des rayons coudés appelée *Bent Pick Ray*.



(a) (b) (c)

FIG. 1.17 – De (a) à (c), des rayons élastiques, des rayons coudés et des rayons déformables [DF02].

1.4.2.3 Bilan intermédiaire

Les techniques d'interaction coopératives peuvent diminuer les temps d'exécution de tâches et rendre des interactions plus naturelles parce que plusieurs utilisateurs peuvent interagir simultanément sur des parties communes d'objets. Mais ces techniques entraînent le problème du passage d'un utilisateur à plusieurs et *vice versa* qui doit se faire de la façon la plus transparente possible. Par ailleurs, il faut montrer en permanence à l'utilisateur que l'action résultante est le fruit de sa propre action combinée à celles des autres utilisateurs. La différence entre l'action qu'il souhaite et celle réellement obtenue doit donc être montrée *via* des métaphores. Enfin, il est important de rappeler qu'une plate-forme logicielle doit limiter le plus possible les latences réseau pour permettre à des interacteurs de se coordonner.

1.4.3 La conscience de soi et des autres

Dans un environnement peuplé par plusieurs utilisateurs, il faut informer chacun des actions que les autres effectuent pour :

- qu'il puisse comprendre grâce à quel moyen des éléments de l'environnement virtuel ont évolué ;
- qu'il puisse prendre la suite d'une interaction.

Ces informations sont encore plus importantes lors d'actions coopératives puisque toute action d'un utilisateur est issue de l'observation de celles des autres.

1.4.3.1 Représentation d'un utilisateur

Permettre à plusieurs utilisateurs d'interagir simultanément sur des objets nécessite de montrer à un utilisateur la présence des autres. Mais, avant de voir les autres, un utilisateur doit lui-même comprendre qu'il est « dans » l'environnement virtuel. Une approche simple permettant de plonger un utilisateur dans un environnement virtuel est l'utilisation d'un *avatar* qui est une représentation d'un utilisateur réaliste ou non et souvent en trois dimensions. Cette représentation peut aller d'une forme très simple et abstraite, une sphère par exemple, à l'hyperréalisme en passant par tous les intermédiaires. Le cas le plus simple [CH93] permet surtout de montrer l'emplacement d'un utilisateur aux autres, ou encore de montrer la direction de son regard (pour exprimer ses intentions, ou afin de se voir soi-même pour envisager la façon d'entreprendre des actions) cette méthode est directement utilisée dans bon nombre de jeux vidéo [III99] et est qualifiée de « vue à la troisième personne », ou encore de « vue subjective ». Il est possible d'affiner un peu cette représentation pour obtenir des visages véhiculant des expressions, néanmoins seul un avatar réaliste [DDS+99], [SE00], pourra couvrir l'ensemble des expressions possibles du visage et permettra, en outre, une communication non verbale véhiculant plus d'informations [BBF+97], [MC98]. Les gestes sont un moyen direct de communication non verbale, la posture exprime l'état d'un utilisateur (sa fatigue par exemple), les contacts du corps avec un autre montrent des signes d'affection ou des liens sociaux, les expressions faciales transmettent des émotions ou donnent des sens différents à des mots prononcés, la représentation réaliste du corps permet de reconnaître une personne réelle et de mieux envisager certaines actions, etc. Par ailleurs, l'incarnation de nombreuses personnes dans un environnement virtuel augmente la sensation de présence, la sensation d'un monde partagé par plusieurs personnes. [GMH04] montrent notamment comment utiliser des agents autonomes pour conserver la présence d'avatars dans un environnement virtuel même lorsque leurs manipulateurs humains se sont absentés.

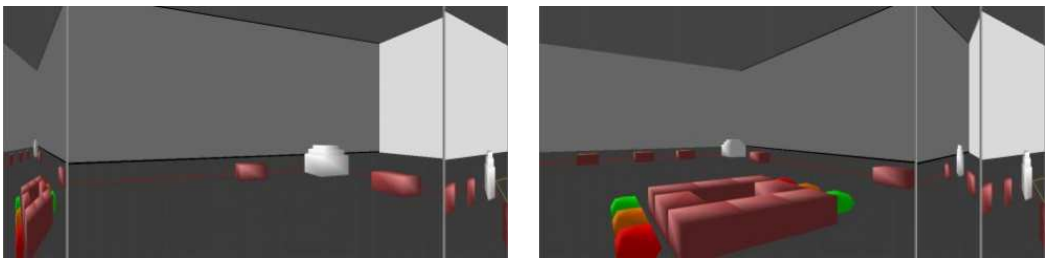
1.4.3.2 Perception des autres

Comment percevoir les autres lorsque le champ de vision est limité par un écran dont la forme réduit la vision périphérique ? De la même façon, comment faire avec des lunettes stéréoscopiques utilisées dans des grands systèmes immersifs, tel qu'un CAVE™ [CNSD+92], qui peuvent partiellement occulter la vue latérale ? Même les systèmes de casques ou de lunettes embarquant des écrans à cristaux liquides ne fournissent généralement pas une vision latérale à grand champ. Il est pourtant indispensable de percevoir les actions des autres et, souvent, cela passe par la vision de ce qu'ils font. Comment également voir ce que les autres regardent lorsque les avatars utilisés ne sont pas très réalistes ou que le système d'affichage n'a pas une finesse de détail suffisante pour bien distinguer la direction du regard ?

Ces problèmes influent directement sur le flot des actions menées au cours d'une interaction [HFH+98]. Dans le monde réel, on peut être amené à aider quelqu'un d'autre parce qu'on a été capable de le percevoir en train d'effectuer une action. Cette situation peut se reproduire dans un environnement virtuel. Cependant, un environnement virtuel inhibe certains sens comme le toucher, en altère d'autres comme la vue ou l'ouïe, et induit le recours quasi systématique à une communication verbale : les gens ont besoin de se dire ce qu'ils font en plus de simplement le faire. Il faut donc des solutions pour compenser les problèmes suscités par les environnements virtuels.

Perception des autres par la vue. L'amélioration de la perception des autres par la vue peut passer par l'extension du champ de vision, la focalisation de la vue, ou encore la matérialisation du champ de vision. La première méthode est très adaptée au cas des systèmes fournissant un champ de vision restreint, les autres sont plus générales. Enfin, la vue permet aussi de se rendre compte qu'un objet a été sélectionné ou est en cours de manipulation.

[FBHH99] proposent d'étendre le champ de vision par le biais d'une métaphore exploitant l'idée de lentilles périphériques pour la vue. Ils partent du constat que le champ de vision dans un environnement virtuel est souvent limité au tiers des capacités humaines. Dans ces conditions, le champ de vision se réduit donc à 50–60 degrés. Cette situation ne vient pas directement de l'environnement virtuel mais de l'écran qui peut créer des distorsions en affichant des images aussi larges. La figure 1.18 (a), illustre l'utilisation des lentilles où la vue centrale est favorisée tandis que les vues latérales sont présentes mais compressées latéralement. Ces lentilles permettent surtout de pouvoir jeter des coups d'œils latéraux. Lorsque l'utilisateur souhaite voir correctement en périphérie, il peut basculer une vue secondaire en principale. La vue centrale devient alors distordue tandis que la vue secondaire choisie (la vue de gauche sur la figure 1.18 (b)) devient la vue principale.



(a) (b)

Fig. 1.18 – Illustration de l'utilisation des lentilles périphériques. (a) La vue centrale est placée comme vue principale. (b) C'est au tour de la vue de gauche [FBHH99].

À l'inverse d'une extension du champ de vision, il pourrait être envisagé de diriger le regard d'un utilisateur vers des points où d'autres utilisateurs interagissent.

Par exemple, des altérations de l'image (renforcements de contrastes, modification de l'éclairage, etc.) peuvent ainsi diriger le regard [BDDG03].

Alors que les deux méthodes précédentes travaillent sur le champ de vision à travers lequel un utilisateur voit le monde, on peut chercher à montrer aux autres utilisateurs le champ de vision d'un utilisateur en matérialisant sa pyramide de vue par des fils de fer (cf. figure 1.19) même si, en pratique, les utilisateurs ne se servent des fils de fer de la pyramide que pour repérer plus facilement la position d'un utilisateur [FBHH99].

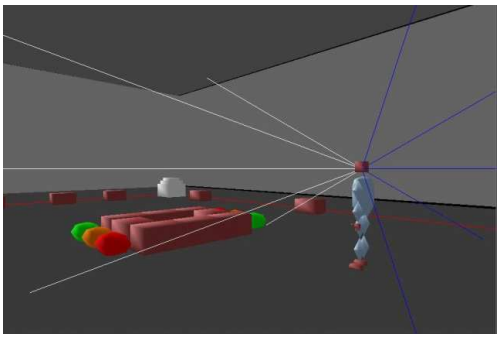
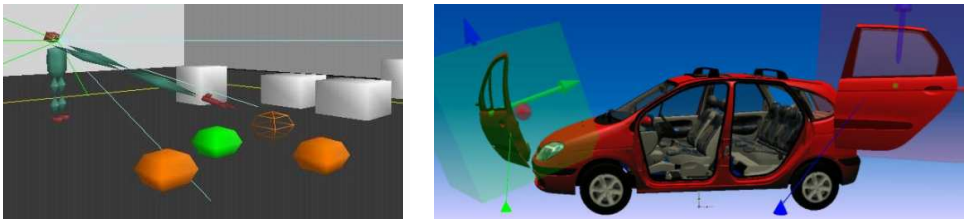


FIG. 1.19 – Illustration de la délimitation du champ de vision d'un interacteur par des fils de fer : les fils clairs délimitent la zone non distordue de la vue et les fils foncés celles des lentilles périphériques [FBHH99].

La pyramide de vue, explicitée précédemment, ne permet pas de montrer qu'un utilisateur est en train de regarder un objet si cet utilisateur est en dehors du champ de vision du premier utilisateur. Cette situation est tout à fait commune dans le monde réel mais sa fréquence d'apparition est augmentée dans un environnement virtuel si le champ de vision est réduit. Certes les lentilles périphériques permettent de voir partiellement sur les côtés mais la distorsion qu'elles entraînent peut empêcher de se rendre compte qu'un utilisateur fixe un objet en particulier. Il faut alors un moyen indiquant clairement qu'un objet est sélectionné et par quel interacteur. Une solution consiste à modifier la géométrie ou l'apparence d'un objet lorsqu'il est sélectionné. [FBHH99] basculent ainsi la géométrie d'un objet en fils de fer (cf. figure 1.20) et étirent le bras de l'avatar de l'utilisateur jusqu'à l'objet en cours de manipulation. L'article [DLT04] donne plusieurs façons de montrer la sélection des portières d'une voiture, par exemple par le biais de volumes englobants transparents de couleurs différentes (cf. figure 1.20). Dans [DF09], les auteurs choisissent de permettre aux utilisateurs de sélectionner des objets virtuels à l'aide d'un pointeur 2D qui semble rester à la surface de leur écran. En pratique, ce pointeur 2D est en fait la projection d'un rayon virtuel 3D perceptible par les autres utilisateurs qui partagent le même univers virtuel. L'objectif est de tirer profit des deux techniques de manipulation. Un rayon virtuel permet d'indiquer précisément un objet non occulté distant ou proche, comme dans [DFNA08] par exemple. Quant à un pointeur 2D, il facilite la manipulation des rotations d'un objet par rapport à la manipulation des rotations via un rayon virtuel.



(a) (b)

FIG. 1.20 – Illustration d'un objet sélectionné : il apparaît en fils de fer et se trouve au bout des bras de l'avatar de l'interacteur [FBHH99]. Sur (a), la sélection est montrée *via* des boîtes englobantes [DLT04].

Perception des autres par les sons. Prenons deux origines de sons : la parole et des bruits, ceux dû à des entrechoquements de pièces par exemple. La parole est un moyen simple de communication et l'un des plus naturels. De plus, grâce à sa mise en œuvre relativement aisée, plusieurs plates-formes de réalité virtuelle permettent l'utilisation de la voix : MASSIVE [GB95], Spin3D [DDS+99], etc. La voix permet aussi à des utilisateurs de compenser une mauvaise perception de l'environnement virtuel. Par exemple, des utilisateurs placés dans un monde partagé peuvent discuter à propos d'un objet virtuel qu'ils voient simultanément sous des points de vues légèrement différents et, ainsi, compléter leurs informations mutuellement [HFH+98]. Malheureusement, la voix entraîne des discontinuités lors de coopérations [BPO96], notamment lorsqu'un utilisateur lance un appel aux autres pour démarrer une action mais finalement en arrive à agir lui-même par défaut. C'est à ce moment, où l'on attend une réponse de la part de quelqu'un, qu'un délai est introduit. De même quand un utilisateur veut s'assurer que personne n'a l'intention de faire ce qu'il souhaite effectuer, il lance un appel à tout le monde et attend des réponses avant de se décider à agir. Enfin, la voix devient difficile à distinguer lorsque plusieurs utilisateurs parlent en même temps ou lorsque les micros sont dans un environnement bruité. Par contre, lorsque deux personnes parlent ensemble pour se dire ce qu'elles font ou souhaitent faire, le dialogue est fluide.

En dehors de la parole, le bruit produit par l'entrechoquement de pièces ou d'outils lors d'une interaction est un indicateur des événements importants pour l'utilisateur et ceux qui l'assistent lors de la réalisation d'une tâche (cf. section 1.2.2.3).

Perception des autres par l'haptique. Lorsque plusieurs interacteurs maintiennent ou « touchent » un objet en même temps, son déplacement par l'un des interacteurs va provoquer un retour de force chez les autres interacteurs, ou encore une vibration. L'inconvénient avec un système à retour de force est qu'il est difficile de faire la différence, dans certaines situations, entre le retour de force dû à des collisions et celui dû à l'action opposée d'un autre interacteur. Néanmoins, cette modalité introduit une information kinesthésique fondamentale de l'activité des autres participants.

1.4.3.3 Bilan intermédiaire

La conscience de soi et de celle des autres au sein d'un environnement virtuel passe, au minimum, par une représentation physique de l'utilisateur — avec une représentation de ses outils d'interaction éventuellement — et une information de position. L'avatar élaboré permet des communications plus riches car non verbales. La perception des autres peut se faire *via* différents canaux sensoriels : la vue, l'ouïe et le toucher. Deux problèmes récurrents sont :

- ne pas « noyer » les utilisateurs par un flot trop important d'informations ;
- permettre à l'utilisateur de faire la différence entre les résultats de ses seules actions et les résultats de combinaisons d'interactions de plusieurs utilisateurs.

1.5 Les fondements techniques

Nous allons parler à présent des solutions techniques permettant de mettre en œuvre la coopération en environnement virtuel : comment faire en sorte que leurs utilisateurs situés potentiellement à des endroits géographiques différents puissent partager un même monde simultanément et y agir ? Nous verrons que la résolution de ce problème passe par le choix d'une architecture logicielle et d'algorithmes réseau.

1.5.1 Un principe pour le logiciel : le découplage

En 1993, Shaw et *al.* [SGLS93] ont proposé un modèle découplé de simulation constitué de plusieurs grandes parties. Notamment, une partie, nommée « *Computation* » s'occupait du calcul de la simulation. Elle tirait ses données d'une partie nommée « *Interaction* ». Les résultats des calculs pouvaient modifier des modèles géométriques par la partie « *Geometric model* », qui, eux, étaient ensuite affichés à l'utilisateur par la partie « *Presentation* ». En fonction de ce qu'il perçoit, l'utilisateur agit par le biais de la partie « Interaction ». Ce découpage présente deux intérêts majeurs :

- il permet de découpler les différentes parties pour ne pas que les faibles performances de certaines en limitent d'autres. Le but final est d'obtenir une animation fluide pour l'utilisateur de l'environnement virtuel. Ainsi, chaque partie est susceptible de pouvoir être exécutée à une fréquence différente des autres parties. Par exemple, un affichage stéréoscopique[21] peut être conservé à une vitesse de 60 Hz pour chaque œil alors que le périphérique de position employé ne fonctionne qu'à 20 Hz ;
- il fournit un guide pour le développement d'une application de réalité virtuelle.

La mise en œuvre du multi-fréquentiel passe souvent par deux moyens : l'usage de processus lourds ou de processus légers[22]. Une autre méthode consiste à travailler sur l'ordonnancement d'objets qui font évoluer l'état de la plate-forme logicielle. Par exemple, prenons le cas des *objets de simulation* dans OpenMASK [MAC+02]. Un objet

[21] En fait, l'article ne précise pas le mode d'affichage utilisé. Il s'agit, avant tout, de faire comprendre que l'affichage peut avoir une fréquence très différente du reste de l'application.

[22] En anglais, « thread ».

de simulation constitue la brique élémentaire des simulations virtuelles sur cette plate-forme. Chacun de ces objets peut être activé à sa propre fréquence pour faire évoluer la simulation virtuelle, ainsi il est par exemple possible de faire exécuter un objet deux fois plus souvent qu'un autre. En jouant sur des fréquences différentes, il est possible d'obtenir des animations qui paraissent fluides aux utilisateurs tout en évitant d'activer inutilement des objets (ceux qui pilotent des périphériques matériels par exemple).

1.5.1.1 Abstraction du matériel

L'abstraction du matériel est une technique indispensable pour concevoir et implémenter du code source qui ne soit pas dépendant de matériel particulier. Il devient alors possible de changer aisément de matériel sans avoir à effectuer de profondes modifications du code existant. L'avantage est de pouvoir suivre les évolutions du matériel en le renouvelant. Une autre raison est de pouvoir choisir le matériel le plus adapté à la situation. Enfin, il devient possible de développer des applications qui fonctionnent dans différents lieux ayant des configurations matérielles légèrement différentes.

L'amélioration de la portabilité d'une application de réalité virtuelle pour différents types de matériels est un objectif de longue date. Dans le but d'obtenir un système multi-fréquentiel, MR Toolkit [SGLS93] propose une collection de processus[23] qui peuvent se ranger sous trois couches. La première est celle du matériel. Chaque processus peut communiquer avec le reste de MR Toolkit par des sockets. Pour gérer un nouveau matériel, il faut alors créer un nouveau programme ou modifier un programme existant. Au-dessus de cette couche en apparaît une autre : elle est composée d'une collection de paquetages qui fournissent une interface haut-niveau de la couche matérielle (si un paquetage correspond à un périphérique matériel alors il est connecté au processus correspondant de la couche inférieure) et fournissent un moyen d'utiliser/construire des techniques d'interaction. Enfin, une couche finale représente l'utilisation des deux couches inférieures. Elle contient, par exemple, la configuration et le code qui font fonctionner l'application.

GNU/MAVERIK [HCK+99] utilise des « callbacks[24] ». Par exemple, pour une souris, trois sont utilisées. L'une est appelée pour scruter des nouvelles positions. Une deuxième convertit les coordonnées obtenues dans l'espace 3D de l'environnement virtuel. La dernière callback réagit à l'appui de boutons. Les développeurs ont la charge de faire remonter les données obtenues aux autres parties de l'application (pour du rendu graphique ou des détections de collisions notamment).

VR Juggler [BJH+01] est programmé en C++ et repose sur un mécanisme d'héritage de classes. Les périphériques sont divisés en plusieurs catégories (périphériques de positions, périphériques analogiques, périphériques discrets, etc.), chacune ayant une interface C++ correspondante. Implémenter un nouveau matériel consiste à implémenter les interfaces adaptées. Il suffit ensuite d'appeler la méthode adéquate pour, par exemple, obtenir une position qu'un périphérique peut fournir. DIVERSE [KAKS02] est

[23] Au sens Unix.

[24] Cette plate-forme est écrite en C pour permettre de changer facilement des pointeurs de fonctions et, ainsi, des comportements.

une autre plate-forme de réalité virtuelle qui reprend cette idée de hiérarchie de classes, en C++, représentant des périphériques matériels pour les abstraire. De la même façon, citons également MORGAN [OHL+04, BLO+05] qui est un framework pour la réalité virtuelle et la réalité augmentée.

OpenTracker [RS01] ne semble pas utiliser une hiérarchie de classes pour abstraire des périphériques matériels mais isole chacun d'eux à l'intérieur d'un *nœud* de type *source*. Les éléments de la plate-forme sont organisés autour du principe de flux de données[25] : un *nœud* envoie en continu des données à un autre *nœud*. Un nœud peut être une source, un *filtre*, c'est-à-dire qu'il applique des traitement aux données reçues, ou une terminaison. Dans GASP [DCDK98], puis dans son successeur OpenMASK [MAC+02], l'équivalent d'un nœud est un objet de simulation. De la même façon que dans OpenTracker, un objet de simulation peut renfermer un pilote d'un périphérique matériel [DM00]. Enfin, Nous pouvons aussi citer les travaux de Figueroa et *al.* [FGW01] qui se basent sur les mêmes principes pour l'abstraction du matériel et la circulation des données.

DWARF [BBK+01] est un framework, plutôt destiné à la réalité augmentée, de services distribués réutilisables décrits en XML. Son originalité est que ses services exposent leurs besoins et leurs capacités sur le réseau pour permettre des connexions dynamiques. En comparaison, la plateforme OpenTracker [RS01] et celle de Figueroa et *al.* [FGW01] n'utilisent XML que pour une description au sein de fichiers de configuration. Par exemple, le gestionnaire de service de DWARF connecte le service fournissant l'image capturée par une caméra à un système d'analyse d'images pour du suivi. Ces gestionnaires de services s'exécutent sur différents systèmes (éventuellement différentes machines) et communiquent entre eux *via* une couche de communication basée sur CORBA [COR].

Jusqu'à présent, nous avons donné des méthodes d'abstraction logicielle qui intègrent la couche logicielle d'abstraction à la plate-forme considérée. Une autre méthode consiste à déplacer la couche d'abstraction en dehors des plates-formes de réalité virtuelle pour la rendre indépendante. VRPN[26] [THS+01, VRP] comporte un ensemble de serveurs qui s'exécutent sur les machines reliées à des périphériques de réalité virtuelle, pour envoyer sur un réseau les données acquises à partir d'un périphérique. Il est alors possible d'avoir plusieurs machines lisant les données provenant d'un même périphérique (et donc d'une même machine). VRPN est à considérer comme un fournisseur d'interfaces à un ensemble de fonctions de périphériques : réception de positions / orientations, réception d'évènements pour des boutons, etc. Cette bibliothèque, accompagnée de ses serveurs, a connu un succès important dans le domaine de la réalité virtuelle et la liste des périphériques qu'elle exploite s'est allongée avec les années. Face aux solutions intégrant directement l'abstraction logicielle au sein de la plate-forme de réalité virtuelle, elle offre une réutilisabilité plus importante. Toutefois, l'usage d'un réseau peut introduire des latences, et des variations de latence, qui sont problématiques

[25] En anglais, « dataflow ».
[26] *Virtual-Reality Peripheral Network.*

avec des périphériques fonctionnant à haute fréquence (ex : certains systèmes de suivi optique).

1.5.1.2 Construction de techniques d'interaction

Nous donnons d'abord des références vers des langages décrivant des techniques d'interaction dans des environnements virtuels, puis des moyens d'écrire ces langages.

Langages de description d'environnements virtuels et d'interactions. Plusieurs langages se sont succédés pour décrire des environnements virtuels. Nous nous attachons, en particulier, à leur capacité à décrire des interactions. L'intérêt de ces langages est de permettre des changements dans un environnement virtuel sans avoir à modifier des lignes de code source. Cela permet d'accélérer le développement d'environnements virtuels et d'ouvrir les portes du développement à un public moins spécialiste.

L'un des premiers langages de description d'environnements virtuels, et probablement le plus connu, est VRML [VRM]. Ce langage repose sur un graphe de scène composé de nœuds. Chacun représente les différents éléments de la scène, qu'ils soient issus du monde réel (ex : une sphère, une lumière, etc.) ou conceptuels. Il est possible de générer des événements automatiquement (ex : lorsqu'un utilisateur passe le curseur de la souris sur un objet particulier) par des « *sensor nodes* » qui seront « routés » vers un autre nœud. Il devient possible d'exécuter des scripts (généralement en Javascript ou en Java) et donc d'introduire des comportements complexes dans un environnement virtuel. VRML a ensuite été remplacé par X3D [X3D] qui est écrit en XML [XML] pour être plus extensible et plus facile à manipuler par des programmes (création et édition).

CONTIGRA [DHM02] propose de mieux structurer X3D grâce à une approche de plus haut-niveau. Selon ses auteurs, X3D, comme VRML, ne permet pas l'utilisation d'outils-auteur de haut-niveau pour créer des composants réutilisables et les employer, notamment pour la définition explicite de composants pour des interfaces en 3D. CONTIGRA propose de définir des composants 3D par un graphe de scène comportant plusieurs graphes de scène fils. Un graphe de scène peut définir la géométrie d'un composant 3D, les sons qu'il peut produire, son comportement, etc. Par ailleurs, un graphe peut être réutilisé par plusieurs composants différents.

COLLADA [COL] est un format pivot basé sur XML pour la création de contenus 3D. Son but n'est donc pas d'être distribué dans des applications finales aux utilisateurs[27], comme un jeu vidéo par exemple. Ce format intervient lorsqu'il faut échanger des données entre différents logiciels participant à la chaîne de création de contenus. Par exemple, un logiciel d'ajout d'effets visuels est appliqué aux données produites par un logiciel de modélisation 3D. Sa large utilisation dans l'industrie du jeu vidéo est probablement due à son concepteur, Sony, qui a associé ce format aux développements logiciels sur ses consoles de jeu.

[27] Pour des raisons de performances, les développeurs de logiciels privilégient souvent des formats de fichiers *ad-hoc*. Un tel choix est fréquent pour des jeux vidéos où il faut lire le plus rapidement possible un fichier pour faire apparaître un monde 3D aux joueurs.

Avec les plates-formes basées sur un flux de données, un ensemble de composants connectés (ex : des filtres ou des nœuds) est employé. InTml [FGH02] décrit les liens entre les nœuds, ainsi que diverses propriétés des nœuds (ex : nom, description textuelle, code associé, etc.). Il repose sur une extension du format X3D.

Environnements de programmation. Les premières plates-formes de réalité virtuelle était construites d'une façon monolithique et figée : l'application réalisée était en grande partie écrite pour le cas spécifique d'une simulation virtuelle. Puis, la description d'environnements virtuels s'est déplacée vers des fichiers de configuration : d'abord pour décrire les périphériques matériels employés, puis pour décrire d'une façon de plus en plus abstraite les objets peuplant l'environnement virtuel ainsi que leurs relations.

Actuellement, un consensus semble se dégager autour d'une programmation majoritairement effectuée par des composants graphiques en 2D. L'utilisateur-développeur relie les entrées et les sorties des boîtes 2D par des lignes ou des flèches et programme ainsi grâce à une interface graphique. Une boîte 2D représente, généralement, des scripts ou du code à exécuter. Dragicevic et Fekete [DF04] ont proposé leur usage dans le contexte d'applications en 2D qui peuvent utiliser des périphériques autres que le clavier et la souris. Puis, Unit [OF04] a également exploité cette idée mais en proposant une interface aux éléments en 3D (cf. figure 1.21) et à destination de la construction d'environnements virtuels. MORGAN [BLO+05] utilise une interface en 2D pour créer des interfaces pour la réalité augmentée. Comme autre utilisateur de ce type d'interface (cf. figure 1.22), citons enfin Virtools [Vir] qui est un logiciel commercial pour des environnements virtuels 3D.

1.5.1.3 Bilan intermédiaire

L'objectif d'abstraire le matériel a entraîné un travail important qui se divise en deux approches. Dans un premier cas, l'abstraction logicielle se réalise directement dans les plates-formes de réalité virtuelle. Dans un second cas, l'abstraction logicielle est faite en dehors des plates-formes de réalité virtuelle, c'est le cas de VRPN [VRP]. En elle-même, l'abstraction repose souvent sur une identification des fonctions qu'un périphérique met à disposition (ex : la lecture de positions) et sur la construction d'une hiérarchie orientée objets.

Pour décrire des environnements virtuels, VRML [VRM] puis X3D [X3D] ont été employés. Tous deux contiennent peu de moyens pour abstraire le matériel et des techniques d'interaction sous formes réutilisables. Ces manques ont été la source du développement de CONTIGRA [DHM02] et de InTML [FGH02]. Par ailleurs, le format COLLADA est un format pivot répandu dans l'industrie du jeu vidéo pour la création de contenus.

Enfin, nous avons présenté les principes de la programmation visuelle par des branchements d'entrées et de sorties pour connecter des objets. Les objets, ou « boîtes » 2D dans ce cas, renferment des composants logiciels décrivant des comportements pour des outils d'interaction ou des objets interactifs. Un développeur programme en venant piocher parmi les objets existants ou en en créant des nouveaux.

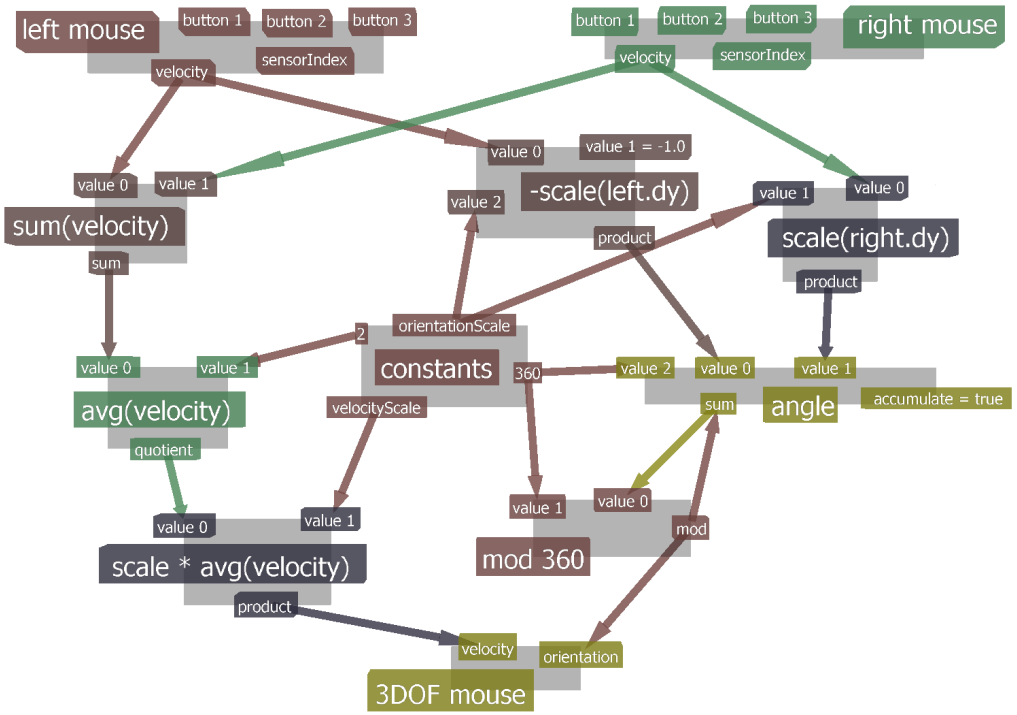


FIG. 1.21 – Ce graphe de flux de données dans Unit [OF04] spécifie le contrôle de six degrés de liberté par l'usage de deux souris. Les molettes des deux souris fournissent le troisième degré de liberté de chacune.

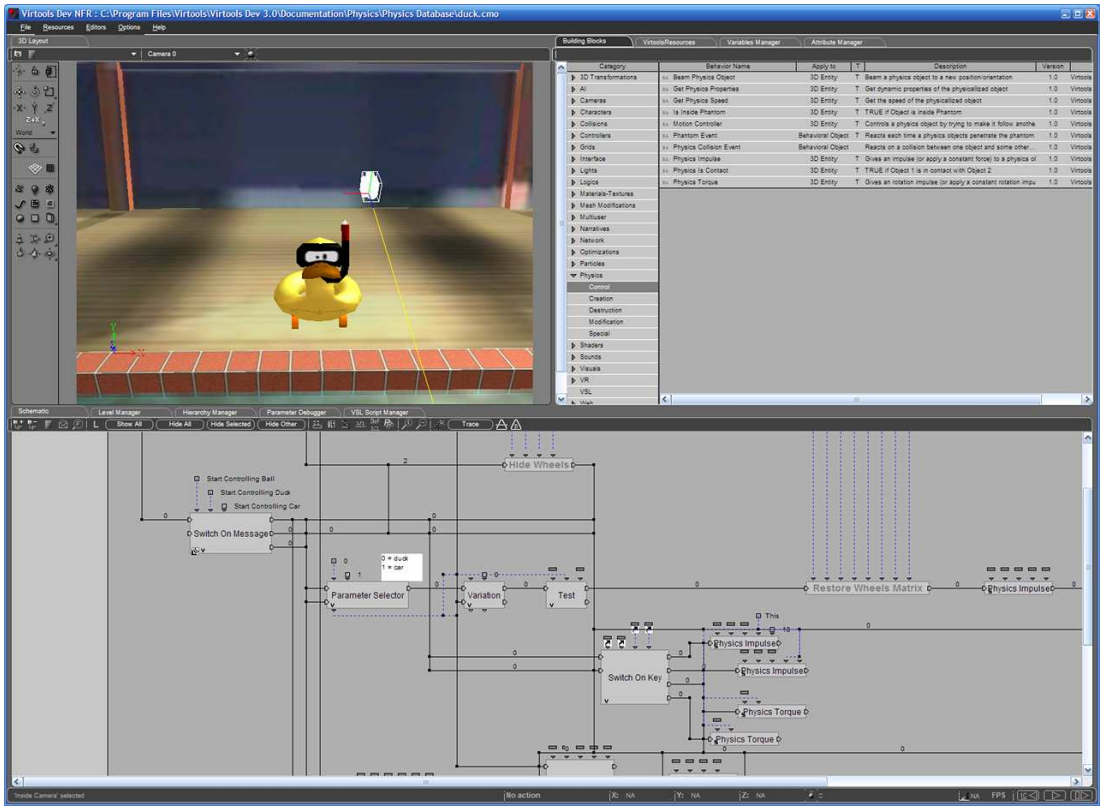


FIG. 1.22 – Illustration de l'interface de Virtools (version 3) [Vir]. Le développeur vient « piocher » des fonctionnalités grâce aux « building blocks » (cadre en haut à droite). Ces blocs peuvent être connectés entre eux, comme illustré par le cadre inférieur de la fenêtre.

1.5.2 Architectures réseau

Les architectures logicielles employées pour la gestion du réseau dans les environnements virtuels peuvent se regrouper sous trois grandes catégories : centralisée, répliquée et hybride. Nous proposons dans cette section d'analyser les qualités et les défauts de chacune de ces catégories. Pour plus d'informations, le lecteur pourra se référer à l'état de l'art présenté en deux parties dans [DWM06a] et [DWM06b], ou encore à celui contenu dans [FDGA10].

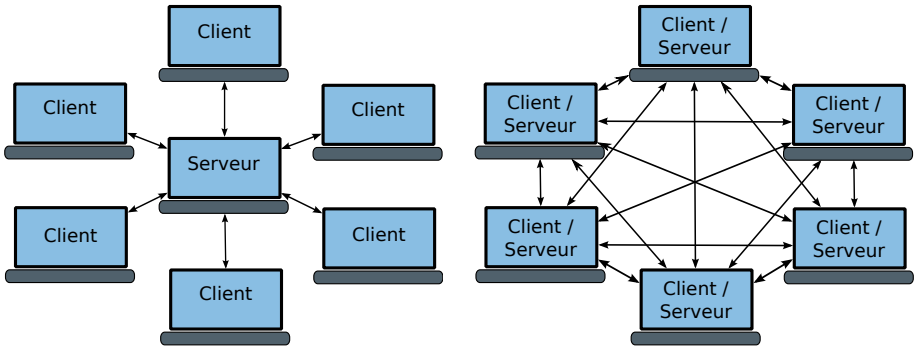


FIG. 1.23 – (a) Illustration d'un réseau centralisé où, de fait, une base de données décrivant l'environnement virtuel est partagée avec l'ensemble des utilisateurs. (b) Illustration d'un réseau distribué. Ces deux illustrations sont basées sur [GlKP94].

1.5.2.1 Architecture centralisée

L'ensemble des données décrivant l'environnement virtuel est stocké sur un seul serveur. Ces données décrivent les objets du monde virtuel par leurs positions dans l'espace, leur état ou encore leurs parties interactives. De leur côté, les sites impliqués dans la simulation interactive ne stockent pas de données descriptives mais disposent généralement, en local, des géométries des objets de l'environnement virtuel pour diminuer le trafic réseau qu'engendrerait l'échange de ce type de données. Dans le cas « extrême » de centralisation, le serveur calcule l'image résultat d'une action à envoyer sur l'environnement de visualisation d'un interacteur. Le serveur calcule tout, et le client affiche le résultat qu'il a demandé au travers d'une action à effectuer. La puissance de calcul graphique du serveur doit alors être très importante.

Lorsqu'un interacteur agit sur un objet, le site où se trouve physiquement l'interacteur envoie au serveur central la demande de mise à jour : une nouvelle position, une nouvelle couleur pour une partie d'un objet, une modification de la géométrie d'un objet, etc. Le serveur a alors à sa charge, d'une part, d'accepter ou de refuser les modifications, et d'autre part de les propager aux autres sites, y compris l'émetteur des commandes, lorsqu'elles ont été acceptées. L'architecture centralisée a ainsi pour avantage la simplicité de son principe de fonctionnement et donc de sa mise en œuvre : chaque site

ne peut recevoir des modifications que d'un seul point, le serveur. De plus, les clients déployés sur chaque site ont un schéma de fonctionnement très simple : toute tentative de modification locale est envoyée au serveur et seul le résultat provenant du serveur est affiché. Le serveur doit donc gérer tout conflit entre différentes actions concurrentes. Sauf si les clients sont à des distances très variables du serveur ou utilisent des liaisons aux performances très différentes, ils reçoivent les résultats d'actions à peu près au même moment. Cette synchronisation conduit à faciliter les interactions coopératives nécessitant une forte coordination. Enfin, cette architecture assure la cohérence entre sites sans avoir recours à des mécanismes complexes à mettre en place. Néanmoins, le fait que toutes les communications passent par le serveur peut le transformer en un goulet d'étranglement s'il reçoit beaucoup de commandes d'interactions de clients ou même s'il doit mettre à jour un grand nombre de clients. Enfin, cette architecture suppose que le serveur doive toujours rester opérationnel pour pouvoir fonctionner, ou que les ruptures de communications doivent rester extrêmement brèves.

1.5.2.2 Réplication de données sur chaque site

La *réplication active* est un modèle de distribution d'objets où l'ensemble des données décrivant l'environnement virtuel est stocké sur chaque site. L'état de chaque site est initialisé à son démarrage (certains sites peuvent rejoindre l'environnement virtuel plus tard au cours de la simulation) et leur évolution est effectuée indépendamment sur chacun. Une plate-forme utilisant cette architecture est Spin-3D [GDGC06] de Orange Labs.

Une interaction sur un site doit communiquer aux autre sites les changements qu'elle a entraînés. Chaque site est donc responsable des interactions qu'il autorise et des calculs des résultats, ces derniers impliquent de fortes charges sur chaque machine. Ainsi, le retour d'informations à un interacteur est immédiat. Malheureusement, des synchronisations sont parfois nécessaires et peuvent s'avérer délicates : comment arbitrer l'accès à un objet par exemple ? Quel interacteur a modifié l'objet en premier ?

De plus, l'interacteur qui agit est le premier à être informé des changements qu'il a produits tandis qu'un autre interacteur qui travaille en coopération avec cette personne ne sera informé des changements que plus tard. Le délai introduit est problématique dans le cas où des interacteurs ont besoin de fortement se coordonner. Néanmoins, ce type d'architecture *pair-à-pair*[28] présente aussi des avantages, notamment celui d'être potentiellement robuste aux pannes parce que la disparition d'un site n'interrompt pas le fonctionnement de l'ensemble de l'environnement virtuel. Toutefois, il devient important de se poser la question de la gestion du cas où un interacteur manipule un objet lorsque son site tombe en panne. Est-ce que l'objet en cours d'interaction doit revenir à l'état qu'il avait avant l'interaction ? Est-ce que l'objet doit rester dans sa position actuelle (par exemple, rester artificiellement en l'air) ? En ce qui concerne la communication, l'envoi de messages par *multicast*[29] (voire par *broadcast*[30]) est néces-

[28] En anglais, « Peer to Peer » (P2P).
[29] Le « multicast » permet d'envoyer des messages à un groupe de machines sur un réseau.
[30] Le « broadcast » permet l'envoi de messages à toutes les machines connectées sur un réseau.

saire pour pouvoir envoyer à chaque site concerné de nouveaux états. Mais ce système a le désavantage de produire une très forte activité réseau et d'être peu exploitable sur des réseaux à grande échelle (ex : l'internet) d'où le recours à des solutions logicielles qui donnent l'illusion d'un « multicast » à un environnement virtuel.

1.5.2.3 Architecture hybride

L'architecture répliquée représente un mieux par rapport à l'architecture centralisée du point de vue de la robustesse aux pannes, de la vitesse du retour d'informations à l'interacteur et des possibilités de passage à l'échelle. Cependant, l'activité réseau est très forte et des problèmes importants d'arbitrage dans les interactions simultanées subsistent. Pour ces raisons, des architectures hybrides ont été développées.

Réplication à la demande. Pour diminuer les échanges réseaux entre sites, tels que ceux d'un système de réplication active, il est préférable d'effectuer l'envoi de messages par *multicast*. Un site ne possède donc que les données qui lui importent ; pour cette raison, ce modèle de distribution d'objets se nomme *réplication à la demande*.

Clients/serveurs et partitionnement de l'environnement virtuel. La base de données décrivant l'environnement virtuel peut être distribuée sur plusieurs sites. Toutes les communications passent par un serveur qui sait où se situent les objets qu'un interacteur peut souhaiter utiliser. Ce système est destiné à afficher plusieurs vues à plusieurs groupes d'utilisateurs tout en ayant un grand nombre d'objets dans l'environnement virtuel. En pratique, cette architecture entraîne une très forte activité réseau concentrée sur le serveur (exactement comme dans le cas de l'architecture centralisée qui n'est autre qu'une variation de l'architecture de celle-ci). Pour contrer ce problème, il est possible de disposer de plusieurs serveurs avec, de fait, des difficultés pour les synchroniser entre eux. Ce cas est exploité par la plate-forme RING [Fun95].

Référentiels et miroirs. Chaque site peut se voir doté de certains objets de l'environnement virtuel, qu'ils soient visuels ou non. Lorsqu'un site possède un objet, cet objet est un référentiel. Le référentiel est chargé de faire évoluer l'objet après toute demande de modification d'une caractéristique de cet objet (position, forme, couleur, etc.). Lorsqu'un interacteur présent sur un site distant essaie d'interagir avec un objet, il passe par un *miroir*, appelé aussi proxy ou fantôme. Un miroir reçoit régulièrement les mises à jour d'un référentiel et les passe systématiquement aux objets (s'il y en a) qui lui sont connectés. L'utilisation d'un miroir diminue les activités réseau parce que la seule communication qui a lieu est celle alimentant le miroir. En effet, les autres objets présents sur le même processus qu'un miroir ne seront en liaison qu'avec ce miroir et non pas avec son référentiel distant. D'une certaine façon, les référentiels jouent le rôle de serveur tandis que les miroirs jouent celui de client. Ce modèle est exploité par OpenMASK [MAC+02].

Migration. Plutôt que de placer statiquement et de façon arbitraire un objet sur un site (par exemple le site où il y a l'interacteur susceptible d'interagir le plus, ou le site le moins chargé au moment de la création de l'objet), il peut être préférable de le placer dynamiquement en fonction d'autres critères, notamment la charge d'une machine au cours de l'évolution de l'environnement virtuel. Par exemple, il devient ainsi possible d'éviter d'avantager un outil par rapport à un autre au niveau de la vitesse de la communication. Lorsqu'au cours de l'utilisation de l'environnement virtuel, un objet peut être déplacé d'un site vers un autre : on parle de *migration* d'objets. Des travaux ont été réalisés dans ce sens sur OpenMASK [DeZ06]. Notons que la géométrie des objets, si elle est déjà présente sur chaque site, n'est généralement pas migrée pour éviter une charge trop importante du réseau.

1.5.2.4 Bilan intermédiaire

Aucune architecture ne présente une solution universelle au problème de la cohérence entre sites. En effet, à l'exception de délais relativement maîtrisés autorisant donc des interactions coopératives fortement coordonnées, l'architecture centralisée présente trop d'inconvénients majeurs pour être intéressante (faible robustesse, puissance de calcul requise du serveur considérable, difficultés de passage à l'échelle, etc.). En utilisant des architectures hybrides, le modèle décentralisé est un meilleur choix même s'il est plus complexe : il offre des interactions avec un retour rapide d'informations (ce qui est très profitable aux interactions coopératives) ainsi que des arbitrages entre interactions simultanées, il utilise raisonnablement le réseau et permet des passages à l'échelle plus aisés.

1.5.3 Algorithmes réseau

Parmi les nombreuses difficultés introduites par les architectures distribuées, notamment hybrides, nous choisissons d'en voir deux qui nous semblent parmi les plus essentielles : la communication entre sites et le maintien de la cohérence entre sites.

1.5.3.1 Communication entre sites

En un même lieu, la communication entre deux entités est immédiate. En distribué, le réseau influe sur la qualité de cette communication en augmentant les délais, voire en perdant des données[31]. Une activité réseau trop importante augmente ces deux aléas. Deux formes de communication entre sites sont utilisables : l'événement et le flot de données. L'événement peut véhiculer une valeur qui est, bien souvent, une valeur ponctuelle : par exemple, un booléen qui passe de faux à vrai et *vice versa*. L'envoi peut souffrir des problèmes décrits précédemment : pertes de données et retards de livraison trop importants. Le flot de données consiste à envoyer des informations en continu. De cette façon, un récepteur trouve toujours une donnée à exploiter, même si cette dernière

[31] La perte de données dépend surtout du protocole de communication sous-jacent employé. Ainsi TCP est un protocole assurant la livraison des paquets réseau, tandis que UDP est plus rapide mais non fiable.

n'est pas forcément la plus à jour. Cette méthode est peu adaptée à des changements de valeurs ponctuels. Par ailleurs, la charge réseau induite par ce système est importante. Ainsi, un mélange entre flots, pour les positions d'objets en mouvements par exemple, et d'événements, pour des communications ponctuelles, peut être envisagé comme dans OpenMASK. Enfin, des communications par appel de fonctions distantes peuvent être employées, selon le modèle CORBA [COR] par exemple.

1.5.3.2 Cohérence entre sites

Dans le cas d'une architecture centralisée, le calcul de l'évolution de chaque entité s'effectue sur une même machine ; les seules difficultés techniques proviennent des entrées concurrentes qu'il faut arbitrer/combiner. En distribué répliqué, chaque entité évolue indépendamment, mais l'état de chaque entité doit rester le même sur chaque site au fil du temps ; cela est assuré par le calcul de l'évolution de chaque entité dont l'avancement est basé sur les tics d'une horloge globale. En distribué hybride, deux cas sont possibles : soit un miroir fait suivre les valeurs des sorties d'un référentiel (voir OpenMASK), soit un miroir est un objet qui calcule par lui-même les valeurs de sortie tant que ses propres calculs ne s'écartent pas trop des valeurs calculées par le référentiel (voir NPSNET [MZP+94]). Cette technique s'appelle le *dead-reckoning* [GlKP94].

La technique du « dead-reckoning » a été introduite dans le protocole DIS (Distributed Interactive Simulation). Ce protocole est la base de la communication réseau de la plate-forme de réalité virtuelle SIMNET [MT95]. Développé à des fins militaires, il est aujourd'hui une norme IEEE (n°1278). Le « dead-reckoning » a deux buts : réduire l'activité réseau en envoyant moins de paquets et maintenir un système en marche pendant des petites coupures réseau. Pour décrire son fonctionnement, nous ne traiterons que des positions. Toutefois, le « dead-reckoning » peut s'utiliser pour tous types de valeurs qui ne présentent pas fréquemment des changements brusques. Son fonctionnement est comme suit. Le référentiel transmet aux autres sites la position de départ d'une entité et des données pour que chaque site puisse la faire évoluer. Chaque site calcule l'évolution de l'entité en se basant sur un algorithme de prédiction que tous les sites et le référentiel connaissent. Typiquement, une position est déduite à partir des positions précédentes permettant d'estimer sa variabilité. Le type de l'algorithme de prédiction à employer peut être envoyé par le référentiel comme dans le protocole DIS. Comme le référentiel calcule des positions par l'algorithme « normal » et par celui de prédiction, il est capable de détecter si les positions calculées par les deux méthodes sont trop différentes. Dans ce cas, une nouvelle position de départ est envoyée à chaque site pour qu'ils réinitialisent leurs calculs respectifs. Si les variations de positions sont peu prévisibles, il faut réinitialiser fréquemment les sites distants, ce qui entraîne une communication réseau importante qui peut être proche de celle obtenue sans « dead-reckoning ».

1.5.3.3 Bilan intermédiaire

Nous venons de voir comment appréhender deux difficultés majeures des architectures distribuées : la communication entre sites et la cohérence entre sites.

La communication peut passer par des flots de données ou des événements. Un événement est une donnée ponctuelle et peut donc être perdue sur un réseau qui n'est pas sûr. Pour des données devant être souvent mises à jour, les flots de données sont plus appropriés : en cas de brèves pertes de données, un récepteur a plus de chances de réussir à obtenir une valeur (mais pas forcément la plus à jour) que si quelques événements avaient été utilisés parce que l'envoi de données est continu même si une valeur n'a pas évolué.

Le « dead-reckoning » peut s'utiliser dans le cas de la communication par flots, voire à événements, pour diminuer l'activité réseau et diminuer l'impact de brèves coupures réseau. Cette technique s'applique dans les cas où une valeur peut être prédite à partir de valeurs précédentes. Son efficacité est très réduite dans le cas de changement de valeurs brusques fréquents. En effet, il faut alors resynchroniser fréquemment les sites distants avec le référentiel, ce qui augmente l'activité réseau.

1.6 Bilan

Au court de cet état de l'art, des techniques usuelles d'interaction mono-utilisateurs ont été présentées. Puis, des techniques à deux mains ont constitué une étape vers les techniques d'interaction collaboratives.

Les techniques d'interaction qui ont été présentées sont relatives à la réalité virtuelle et peuvent être implémentées sur des plates-formes logicielles dont nous avons exposé les fondements techniques à la fois pour l'abstraction du matériel et pour la description de techniques d'interaction. Nous avons aussi présenté les architectures et les algorithmes pour les réseaux afin que ces plates-formes permettent des interactions distribuées.

Dans le chapitre 2, nous nous proposons de faire une analyse de cet état de l'art et de proposer des améliorations dans le domaine des techniques d'interaction collaboratives et des fondements techniques aux plates-formes de réalité virtuelle.

Chapitre 2

Analyse de l'existant et propositions

L'analyse de l'existant présentée dans ce chapitre comporte deux parties dont l'ordre est celui utilisé dans l'état de l'art (chapitre 1). La première partie est une discussion portant sur les techniques d'interaction collaboratives. La deuxième partie discute de la mise en œuvre de techniques d'interaction.

2.1 Discussion à propos des techniques d'interaction collaboratives

2.1.1 Situation

2.1.1.1 Intégration d'actions motrices

Nous avons vu l'émergence de techniques d'interaction collaboratives dans les environnements virtuels 3D. Ruddle et *al.* [RSJ02] identifient trois façons d'intégrer des actions provenant de plusieurs interacteurs pour déplacer un objet virtuel qu'ils partagent.

Séparation. Elle distribue les degrés de liberté d'un objet à manipuler aux interacteurs (ex : Pinho et *al.* [PBF02]). Cette technique s'avère utile dans des environnements virtuels pour permettre aux utilisateurs de se placer à différents endroits afin d'avoir des points de vue différents et des actions différentes. Mais il paraît plus difficile de la considérer comme très correspondante aux interactions produites dans le monde réel. En effet, la séparation des degrés de liberté semble supposer que certains degrés de liberté de l'objet restent figés : par exemple, pour n'effectuer que des translations, il faut que l'objet réel ne puisse pas tourner.

Interpolation. Cette technique est probablement la plus largement implémentée. Elle consiste à interpoler des données. Pour des actions motrices, il s'agit plus particulièrement d'interpoler des positions et des orientations. Elle se retrouve avec le « Bent

Pick Ray » [RHWF06], la manipulation de poutres pour la construction d'un belvédère [RWOS03b, GMMG08], ou le déplacement d'un pare-brise de voiture [SJF09]. À l'exception du « Bent Pick Ray », des mains virtuelles sont généralement employées pour interagir. Toutefois, l'usage des mains est limité à une seule main par personne, ce qui rend l'interaction moins naturelle que ce qu'elle pourrait être. Avec SkeweR [DLT06], l'orientation d'un objet manipulé est obtenue par deux positions uniquement mais l'usage de la rotation autour de l'axe reliant les deux positions est peu naturel.

Intersection. Seule la partie commune entre les positions et les orientations fournies par les interacteurs est employée. Elle paraît être complètement dissociée des manipulations du monde réel parce que, pour la réaliser, il faudrait une évaluation de chaque action souhaitée avant de l'effectuer. À notre connaissance, seul l'article [RSJ02] mentionne cette méthode.

2.1.1.2 Gestion de la prise de conscience

Nous avons vu que le retour d'informations à l'utilisateur est essentiel. Le retour lui permet de comprendre ce qu'il peut faire avec un objet (ex : affichage des déplacements possibles ou zones saisissables d'un objet virtuel) ou encore ce qu'il est en train de faire.

Dans le cas d'interactions collaboratives, un utilisateur doit se coordonner avec ses partenaires. Il faut mettre en place des systèmes qui permettent de suivre les actions d'autrui.

Suivi par la vue. Le suivi des actions des partenaires par la modalité vue peut s'effectuer par deux moyens : 1) en ayant un aperçu direct sur leurs avatars virtuels ou sur leurs corps réels ; 2) en ayant une représentation visuelle de leur actions, sous une forme métaphorique par exemple.

Avec le premier cas, souvent, l'utilisateur voit les actions d'un partenaire au moyen d'une fenêtre qui se superpose au contenu de la scène virtuelle. Il y a généralement autant de fenêtres que de partenaires à observer. En conséquence, la scène peut rapidement devenir cachée. Par ailleurs, l'usage de cette technique entraîne une rupture lors d'une manipulation. L'utilisateur s'arrête, regarde ce que fait son partenaire, puis poursuit. Cette méthode est donc à envisager comme un moyen de savoir où en est un partenaire dans le déroulement d'une manipulation : s'il est en avance ou en retard. Elle peut également servir à regarder comment faire avant d'agir. Le WIM [SCP95] peut être utilisé pour voir les actions des autres utilisateurs grâce à l'affichage du monde en miniature ; en revanche, le WIM ne permet pas, a priori, de voir les actions précisément. D'une façon générale, l'usage de l'incrustation d'un aperçu des autres partenaires semble se limiter au cas de visio-conférences. On affiche donc une personne humaine, mais pas son avatar virtuel, en se focalisant sur le visage ou une partie du corps en action.

Avec le deuxième cas, le but n'est plus de montrer la partie de la scène virtuelle sur laquelle un partenaire travaille, ou de voir le partenaire (réel ou virtuel) faire des gestes, mais, uniquement, de faire comprendre ses actions. Par exemple, la courbure

des rayons du « Bent Pick Ray » [RHWF06] indique les mouvements d'un partenaire, les élastiques au bout des mains virtuelles apparaissant lors de la manipulation d'une poutre [GMMG08] indiquent des mouvements mal coordonnés (*i.e.* les mains virtuelles s'éloignent de la poutre virtuelle), le gel des mouvements de cette même poutre montre que les mouvements proposés sont trop incohérents avec la réalité, etc.

Suivi par l'haptique. Si une personne tire un objet réel vers elle, l'autre personne, tenant le même objet réel, sera tirée avec lui. Des périphériques haptiques permettent de retranscrire les efforts passés à un objet virtuel co-manipulé (ex : Noma et *al.* [NM97]) ; un utilisateur agirait au travers d'un bras à retour d'efforts, par exemple.

Il est également possible d'avoir recours à un retour haptique passif créé par la manipulation d'objets réels. Nous pouvons ainsi citer le cas des interfaces tangibles tenues par plusieurs utilisateurs où l'interface sert de moyen de transmission d'efforts entre ses utilisateurs.

Une alternative au retour haptique actif (qu'il soit par le retour d'efforts ou le retour tactile) est le retour pseudo-haptique [Lec09]. Tout en évitant le recours à du matériel pour le retour d'efforts, ce qui est peut être couteux et prendre du temps de développement, le retour pseudo-haptique peut donner à ses utilisateurs la sensation de manipuler des objets ayant des propriétés physiques grâce à la stimulation de canaux sensoriels qui ne permettent pas a priori de percevoir des efforts, par exemple la vue. Associé à une interface tangible et à des retours visuels appropriés, il devient par exemple possible de donner l'impression qu'un objet tenu est lourd parce que les mouvements amples faits par les utilisateurs réels ne sont convertis qu'en mouvements de faible amplitude dans l'environnement virtuel.

Suivi par l'audio. Lors d'interactions mono-utilisateur, des sons peuvent aider un utilisateur à se rendre compte qu'il y a eu un choc entre l'objet qu'il manipule et un autre objet de la scène virtuelle, si le son est produit au moment du choc et s'il semble correspondre au matériau que les objets réels correspondant auraient. Un même son peut être perçu par deux utilisateurs simultanément et, s'il est spatialisé, les aider dans leur manipulation parce qu'ils peuvent évaluer l'emplacement d'une collision.

Il est à noter également qu'une communication vocale peut aider les utilisateurs à se coordonner pendant une manipulation mais la difficulté pour les deux utilisateurs est qu'ils réussissent à bien voir le même objet virtuel sous le même angle de vue pour en discuter efficacement [HFH+98].

2.1.2 Propositions

Nous souhaitons obtenir une technique d'interaction collaborative qui paraisse naturelle à l'utilisateur. Par conséquent, nous choisissons d'utiliser des mains virtuelles afin d'imiter des gestes de préhension des objets virtuels. À présent, il nous faut considérer une méthode d'intégration d'actions simultanées de plusieurs utilisateurs et fournir à ces utilisateurs des retours pertinents pour les aider.

Du point de vue de l'intégration d'actions motrices, une méthode basée sur l'usage de la moyenne semble la plus indiquée parce que, selon nous, c'est ainsi que les interactions résultantes seront les plus proches de la réalité. Nous avons dit que la séparation des degrés de liberté est difficile à mettre en place dans la réalité, et que l'intersection ne correspond pas à une interaction du monde réel. La moyenne, quant à elle, utilise en permanence toutes les actions proposées par les utilisateurs. Malheureusement, elle est toujours employée à partir de deux « sources » d'actions livrant, chacune, une position et une orientation alors que, dans la réalité, il faudrait, pour deux utilisateurs, la présence de 4 sources, c'est-à-dire une par main. De plus, autant la moyenne de positions est intuitive, autant l'interpolation d'orientations en 3D paraît plus difficile à appréhender, surtout avec plus de deux orientations fournies. Ainsi, une technique qui se base uniquement sur des positions et en déduit des orientations, telle que le « SkeweR » [DLT06], paraît appropriée.

En ce qui concerne les retours à l'utilisateur, nous souhaitons obtenir une technique sans retour haptique « actif », c'est-à-dire sans système à retour d'efforts, ni même tactile. Nous faisons ce choix pour éviter le recours à du matériel coûteux et, parfois, difficile à employer. En conséquence, il nous faudra indiquer à l'utilisateur, par d'autres moyens que l'haptique, les collisions avec des objets virtuels et lui faire ressentir les actions de son (ou ses) partenaire(s). Nous chercherons des solutions par des retours visuels et sonores.

À ce stade de notre réflexion, nous avons proposé la technique de manipulation à trois mains exposée au chapitre 4.

Pour aider les utilisateurs à mieux se transmettre des efforts, un retour passif peut être ajouté, par le biais d'une interface tangible par exemple.

Cette autre réflexion a abouti à l'interface tangible reconfigurable présentée au chapitre 5.

2.2 Discussion à propos de la mise en œuvre technique

2.2.1 Situation

Nous avons vu combien les travaux sur l'abstraction du matériel sont nombreux. En effet, l'indépendance au matériel permet de définir des langages de description de techniques d'interaction qui ne dépendent pas d'un matériel particulier mais, éventuellement, d'une catégorie de matériel (ex : ceux qui fournissent une position 3D en continu).

Un développeur d'environnements virtuels peut décrire, dans une application de programmation graphique (ex : Unit [OF04]), des liens entre objets (ex : des périphériques matériels, des composants 3D) par des liaisons graphiques entre des entrées et des sorties. Ces liens sont décrits dans des fichiers de configuration qui sont souvent écrits en XML (comme pour CONTIGRA [DHM02] et InTML [FGH02]).

La programmation graphique qui a été décrite à travers des interfaces 2D n'est que statique parce que le programmeur relie définitivement des objets entre eux ; il reste à décrire l'aspect dynamique, c'est-à-dire comment des objets peuvent réussir à se

connecter. Il y a d'une part une approche bas-niveau qui consiste à exécuter un ensemble d'instructions (ex : un script) en fonction de situations (cas de VRML et de X3D par exemple). L'avantage de cette méthode est qu'elle requiert des étapes de conception très courtes. Malheureusement, elle présente des inconvénients importants : difficultés de maintenance du code et trop forte dépendance à l'environnement pour lequel le code a été écrit. L'approche complètement opposée consiste à décrire les interactions possibles par une approche de (très) haut-niveau. Elle permet d'écrire dans un langage faiblement dépendant des contraintes logicielles sous-jacentes et favorise la réutilisabilité du code. Un autre intérêt majeur est d'introduire des possibilités de prises de décision par le logiciel. Par exemple, dire à un humanoïde qu'il doit saisir un verre posé sur une table laisse plusieurs actions en suspens : comment va-t-il faire pour attraper le verre si son bras n'est pas assez long, quels verres va-t-il pousser pour atteindre le bon, comment va-t-il mettre ses doigts autour du verre, etc. Un ordre de haut niveau permet donc à l'humanoïde de déduire des actions à effectuer au lieu de suivre un ensemble d'instructions bas-niveau qui auraient été ajoutées méticuleusement.

Nous nous intéressons à la description des interactions à un niveau d'abstraction relativement bas : nous souhaitons à la fois nous éloigner du simple lancement de scripts et pouvoir préciser que certains objets servent à l'abstraction du matériel, à la représentation d'un outil d'interaction virtuel ou à la représentation d'objets interactifs. Nous n'allons pas nous intéresser à des aspects cognitifs (par exemple, la décision d'un humanoïde de se déplacer) mais à la présentation d'informations décrivant simplement l'objet (ex : sa position, sa masse, sa forme, sa couleur, etc.) et les possibilités de modification de ses propriétés. Or, il est très rarement expliqué comment, et pourquoi, un outil d'interaction peut, dynamiquement, établir un lien entre ses sorties et les entrées d'un objet virtuel à manipuler. Par exemple, DWARF [BBK+01] expose, sur le réseau, des services décrits en XML. Mais, les étapes pour utiliser cette description ne sont pas données. Par ailleurs, nous n'avons pas trouvé dans la littérature de moyen de décrire, au niveau que nous considérons, ces interactions dans le cas d'interactions collaboratives : ce qui se passe si deux outils essaient de manipuler simultanément un même objet.

2.2.2 Propositions

Nous souhaitons définir un protocole d'interaction entre des outils d'interaction et des objets interactifs. Ce protocole va établir les étapes de connexion, puis les message échangés, entre un outil d'interaction, qui peut abstraire un périphérique matériel ou représenter une notion plus abstraite, et un objet interactif, c'est-à-dire l'objet à manipuler.

Pour se faire, il faudra que l'objet interactif expose publiquement les possibilités qu'il offre aux outils d'interaction. Devant l'hétérogénéité des langages de programmation employés par les plates-formes de réalité virtuelle (C, C++, Java, etc.), il nous faudra créer un protocole ne reposant pas sur des particularités techniques de certains langages de programmation (ex : Java permet des appels distants de méthodes[1] mais pas C++,

[1] En anglais, « Remote Method Invocation » (RMI).

donc nous ne nous reposerons pas sur ce mécanisme).

Par ailleurs, nous souhaitons permettre des manipulations simultanées sur un même objet interactif. Il va donc falloir décrire cette possibilité au niveau des objets interactifs et les rendre capables de gérer cette situation. Ce double objectif sera atteint grâce à la description des interactions collaboratives dans le fichier de configuration de l'environnement virtuel, ainsi qu'à l'utilisation d'une architecture logicielle à base d'extensions dont nous expliciterons la nature. La structure du fichier de configuration peut être de deux natures : soit au format OpenMASK, soit au format COLLADA [COL]. Le premier est présent pour des raisons liées à la compatibilité logicielle (nous souhaitions utiliser les mêmes fichiers de configuration sur des versions d'OpenMASK n'implémentant pas les fonctionnalités COLLADA) alors que le deuxième représente un choix technologique. En effet, nos partenaires du projet Part@ge ont évalué différents formats de fichiers décrivant des environnements 3D[2] dont X3D, 3DXML[3] et COLLADA. Ce dernier a été retenu parce qu'il connaît un succès grandissant sur les logiciels de création de contenu 3D, qu'il a un large support d'industriels du secteur du jeu vidéo (Sony notamment) et qu'il est facilement extensible (il est basé sur XML). De son côté, X3D semble très peu utilisé bien qu'il soit ouvert et qu'il ait été normalisé en 2005. Enfin, 3DXML paraît trop fermé.

L'ensemble de cette réflexion a conduit au protocole et à la description de son implémentation présentés au chapitre 3.

2.3 Plan des contributions

L'état de l'art présenté au chapitre 1, ainsi que ce chapitre 2, ont tous deux parlé d'abord de techniques d'interaction, puis de fondations techniques sur lesquelles les techniques ont été bâties. Il nous a semblé que le lecteur aurait moins de difficultés à aborder ce manuscrit si nous commencions par des concepts visuels relativement faciles à imaginer. Cet ordre est cependant inverse à l'ordre chronologique de cette thèse.

Le protocole d'interaction a d'abord été développé pour ensuite servir de support à l'implémentation de nouvelles techniques d'interaction. Pour cette raison, nous présenterons le protocole d'interaction au chapitre 3, puis les deux nouvelles techniques d'interaction que nous proposons aux chapitres 4 et 5. Le chapitre 6 présentera les méthodes suivies pour implémenter la gestion de la physique d'une façon générique pour les deux techniques d'interaction.

[2] Pas nécessairement des environnements spécifiquement destinés à la réalité virtuelle.

[3] Format propriétaire de Dassault Systèmes utilisé sur ses différents produits logiciels : Catia, 3DVIA Virtools, etc.

Chapitre 3

Un protocole d'échange entre outils et objets virtuels

3.1 Introduction

Le protocole d'interaction présenté dans ce chapitre décrit comment un outil d'interaction communique avec un objet interactif. Un outil d'interaction interroge un objet interactif pour connaître les possibilités qu'il offre, éventuellement les présente à l'utilisateur si ce dernier doit faire des choix, et en prend le contrôle. Ce protocole permet des interactions collaboratives. Il a été l'occasion de définir ce que sont un outil d'interaction et un objet interactif. En outre, une architecture décomposant outils d'interaction et objets interactifs en petites briques logicielles est proposée. Ces briques permettent de créer des objets facilement réutilisables allant de l'abstraction du matériel à la description d'effets visuels ou sonores. Enfin, notre travail nous permet d'introduire des extensions au langage COLLADA [COL] pour la description des propriétés interactives d'objets interactifs.

3.2 Points à adresser

Nous considérons 3 points à adresser pour réaliser le protocole d'interaction que nous proposons [ADA09a], [AD08]. D'une part, nous souhaitons ne pas reposer directement sur du matériel dédié à la réalité virtuelle mais sur son abstraction. Ensuite, nous voulons implémenter notre protocole de façon générique et réutilisable au sein des outils d'interaction et des objets interactifs. Enfin, nous cherchons à intégrer un travail sur des mécanismes génériques pour du retour d'informations aux utilisateurs (humains ou logiciels) de l'environnement virtuel à notre réflexion sur l'architecture des outils d'interaction et des objets interactifs. Nous supposons également que l'environnement virtuel peut être collaboratif.

3.2.1 Abstraction du matériel

Nous expliquerons par la suite comment outils et objets interactifs sont liés. Pour l'instant, nous nous contenterons de dire qu'un outil peut être manipulé par plusieurs périphériques matériels. L'état de l'art du chapitre 1 a présenté, entre autres choses, la métaphore du rayon virtuel qui permet de sélectionner des objets distants et de les déplacer. Ce rayon peut être lui-même déplacé par différents moyens. Par exemple, nous avons utilisé dans différentes circonstances les touches d'un clavier, une souris classique 2D combinées aux touches du clavier, des Wiimotes de Nintendo ou encore un système optique de suivi de mouvements[1]. Une abstraction correcte du matériel permet de changer aisément celui-ci en altérant au minimum la technique d'interaction qui s'appuie sur cette couche d'abstraction matérielle.

3.2.2 Construction de techniques d'interaction

Les outils d'interaction reposent sur une couche d'abstraction du matériel comme nous venons de le voir. À présent, nous allons décrire comment bâtir des outils au-dessus de cette couche, puis comment employer des outils.

3.2.2.1 Généricité nécessaire d'un outil d'interaction

Un outil a besoin de connaître les possibilités d'interaction qu'offre un objet interactif pour les proposer à l'utilisateur ou encore éviter de démarrer une opération impossible à accomplir. Parfois, les interactions possibles sont réalisées « en dur » par les programmeurs, c'est-à-dire qu'un rayon ne pourra, par exemple, que déplacer certains objets préalablement connectés à lui. Le rôle de l'outil se limite alors à filtrer les objets qu'il peut déplacer pour proposer un choix pertinent à l'utilisateur.

Nous souhaitons éviter au maximum tout « précâblage » et reposer sur les propriétés découvertes dynamiquement lors de l'interrogation d'objets interactifs.

3.2.2.2 Instanciation/utilisation d'outils

Une conséquence directe de l'abstraction du matériel est une plus grande facilité d'implémentation de techniques d'interaction.

Avec les plates-formes basées sur un flux de données, un ensemble de composants connectés (ex : des filtres ou des nœuds) est employé. La connexion peut être opérée par une interface graphique où l'utilisateur déplace des « boîtes », qui représentent des filtres (ex : figure 1.21).

La description d'un outil peut s'effectuer de différentes façons au sein d'un fichier. InTml [FGH02] repose sur un ensemble de propriétés interconnectées décrites par une extension du format X3D [X3D]. CONTIGRA [DHM02] propose également un format étendant X3D en employant plusieurs graphes de scène X3D. Les graphes définissent des géométries d'objets, des propriétés audio et des comportements d'objets virtuels.

[1] Dans notre cas, il s'agissait d'un ARTracking de la société A.R.T. [ART].

3.2.3 Prise en compte du retour d'informations à l'utilisateur

Du point de vue des objets interactifs virtuels, les *Smart Objects* [KT98] encapsulent les propriétés, les comportements et les scripts associés à chaque interaction [SVP07]. Un objet interactif dispose ainsi d'informations pertinentes pouvant être renvoyées à l'utilisateur afin de l'aider. D'abord, un utilisateur a besoin de connaître les actions qu'il peut appliquer à un objet interactif (ex : le déplacer, changer sa couleur ou modifier sa forme). Des métaphores visuelles peuvent alors être employées et se trouver combinées à d'autres modalités de retour [SLMA06]. Le retour d'informations aide l'utilisateur à comprendre ce que ses actions entraînent. Dans un environnement virtuel collaboratif, les utilisateurs doivent absolument comprendre les actions effectuées par les autres utilisateurs afin de : les aider, poursuivre leurs actions ou encore effectuer des actions simultanées avec eux [GG99].

3.3 Intérêts de notre approche

Nous souhaitons permettre la distribution géographique de sessions d'interactions collaboratives en environnement virtuel. Les utilisateurs se partagent le même monde virtuel, au même moment, tout en étant sur des sites réels potentiellement différents. Chaque utilisateur ne peut interagir avec un objet interactif virtuel qu'au travers d'un outil d'interaction virtuel.

3.3.1 Intérêts généraux

Les outils et les objets interactifs correspondent à ce que nous avons présenté comme « filtre » ou « nœud ». Notons que les implémentations de notre approche ont été réalisées sur la plate-forme de réalité virtuelle OpenMASK [LCAA08] où seule la notion d'*objet de simulation* est disponible. Nous verrons comment transformer des objets de simulation en outils d'interaction virtuels ou en objets interactifs virtuels. Pour l'instant, assimilons un objet de simulation à un filtre ou un nœud.

OpenMASK gère lui-même la communication bas-niveau inter-objets. Notre protocole de communication entre outils d'interaction virtuels et objets interactifs virtuels est bâti sur la communication fiable apportée par OpenMASK. Le protocole apporte les avantages suivants.

Développement de techniques d'interaction facilité. Un outil d'interaction ne nécessite pas l'écriture de grandes quantités de code très spécifiques pour chaque technique d'interaction. Pour les interactions entre outils d'interaction et objets interactifs, le protocole est construit dans le but d'être un moyen de communication très générique.

Modification dynamique des propriétés des objets interactifs. Une propriété d'un objet interactif peut être rajoutée ou supprimée dynamiquement. Par exemple, un objet interactif peut être non-déplaçable parce qu'il n'offre pas un attribut de `Position`. Cependant, cet objet interactif peut devenir déplaçable si on lui ajoute l'attribut.

Modification dynamique des permissions d'accès à des objets interactifs. Les changements peuvent avoir lieu au moment de l'exécution de l'environnement virtuel.

Interopérabilité entre différentes plates-formes de réalité virtuelle lors de leurs exécutions. L'indépendance de notre protocole de communication vis-à-vis de langages de programmation facilite l'adaptation de plates-formes logicielles existantes à son usage. Par exemple, il n'est pas fait usage des mécanismes d'appels distants de méthodes proposés par Java.

Proposition d'un nouveau protocole de communication pour la communication entre outils d'interaction et objets interactifs. Ce protocole fonctionne indifféremment en contexte non-distribué et distribué. La phase de connexion à un objet interactif est légèrement surchargée parce qu'il faut questionner l'objet et analyser sa réponse au lieu de se connecter simplement à lui (lorsque la liaison possible est connue d'avance). Une fois la connexion effectuée, l'envoi de données s'effectue de la même manière qu'une connexion sans utilisation de notre protocole.

3.3.2 Intérêts pour l'implémentation des outils et objets interactifs : usage d'extensions logicielles

Comme nous l'avons écrit, l'indépendance du protocole vis-à-vis d'un langage de programmation particulier facilite son implémentation sur de multiples plates-formes logicielles existantes. Le protocole a également joué un rôle dans la façon d'implémenter les outils d'interaction virtuels et les objets interactifs virtuels.

L'implémentation du protocole s'est faite à partir d'*extensions logicielles*, qui seront explicitées dans ce chapitre (cf. paragraphe 3.6). L'agrégation d'extensions au sein d'objets virtuels OpenMASK conduit à la création d'outils d'interaction et d'objets interactifs. Les extensions introduisent plusieurs avantages pour les plates-formes de réalité virtuelle.

Amélioration du point de vue ingénierie logicielle. Les modèles proposés dans [FGW01] et [RS01] conduisent à la création de filtres/nœuds d'usage général. Par exemple, s'il est besoin de recourir à un filtre spécifique pour un périphérique matériel, il faut créer un filtre au nom du type *FiltreParticulierPourUnPeripherique* qui ressemblera à un filtre d'usage général alors qu'il n'est prévu que pour un usage en conjonction avec un périphérique matériel précis. Les extensions logicielles sont prévues pour être spécifiquement associées à un outil d'interaction ou à un objet interactif pour établir un lien fort entre eux : l'extension constitue une partie d'un outil d'interaction ou d'un objet interactif.

Amélioration des performances. Nos applications peuvent être multi-sites donc la répartition de charges peut parfois améliorer les performances générales d'une session. Du fait de l'encapsulation des extensions à l'intérieur des outils d'interaction et des

objets interactifs, la distribution des outils d'interaction et des objets interactifs est facilitée. Par exemple, faire migrer un outil sur la machine faisant évoluer l'objet interactif manipulé diminue la latence entre ceux-ci et offre une interaction plus agréable à l'utilisateur. Avec les filtres/nœuds, la migration est plus complexe parce qu'elle nécessite le déplacement d'un ensemble de filtres/nœuds qu'il faut avoir identifiés comme constituant un outil d'interaction. Pour des filtres/nœuds, il faut donc utiliser une structure de données qui permet de connaître les filtres/nœuds qui constituent un outil d'interaction. Avec les extensions, une structure de données supplémentaire est superflue parce que le lien entre un outil d'interaction et ses extensions est toujours défini.

Amélioration de l'adaptabilité des outils d'interaction et des objets interactifs. Il est possible d'ajouter des fonctionnalités dynamiquement à un outil d'interaction ou à un objet interactif en chargeant de nouvelles extensions. Par exemple, un objet interactif très simple ne sachant effectuer que des translations pourrait apprendre à modifier aussi son orientation grâce à une nouvelle extension. Il est également possible d'enlever des extensions pour supprimer des fonctionnalités au cours de la simulation virtuelle.

3.4 Le modèle : outils d'interaction et objets interactifs

Une interaction est vue comme un dialogue entre un outil d'interaction et un objet interactif. L'outil d'interaction envoie des données à un objet interactif qui est responsable et libre de leur interprétation. L'outil d'interaction a la charge de suivre l'évolution de l'objet interactif qu'il manipule. Un suivi par l'outil d'interaction peut lui permettre d'adapter son propre comportement à l'évolution de l'objet interactif. Par exemple, un objet interactif n'aura probablement pas le même déplacement s'il est déplacé par un seul interacteur ou par plusieurs simultanément.

3.4.1 Outils d'interaction

Un objet interactif ne peut être manipulé qu'au travers d'un outil d'interaction. Un tel outil correspond à ceux décrits dans l'état de l'art, au chapitre 1, comme une main virtuelle ou un rayon laser virtuel. Ces outils d'interaction peuvent être manipulés par des outils matériels, c'est-à-dire par des périphériques matériels (clavier, souris 2D, souris 3D, gant de données, système de suivi optique, etc.) ou encore par un système entièrement logiciel.

Un outil d'interaction est un élément logiciel permettant de proposer à un objet interactif des modifications de ses attributs. Par conséquent, un outil est composé d'attributs. Chaque attribut possède un nom unique (au niveau de l'outil), contient une donnée typée, est membre d'une catégorie et est caractérisé par un niveau d'accès. Par exemple, un outil peut contenir un attribut `Position` qui contient une donnée de type

`Transform`[2] et a un niveau d'accès suffisamment élevé pour l'autoriser à modifier n'importe quel objet interactif de l'environnement virtuel. Cet attribut ne représente pas la position actuelle de l'outil mais celle qu'il souhaite appliquer à un objet interactif. Quant à la catégorie, elle est une chaîne de caractères permettant d'avoir un supplément d'informations concernant la nature du contenu de l'attribut (ex : c'est une position dans le repère global, ou bien c'est une position relative à un repère local). L'usage de ce champ sera mis plus avant dans le paragraphe mettant en relation outils d'interaction et objets interactifs virtuels.

L'outil d'interaction devra gérer l'envoi de la valeur proposée par chaque attribut à chaque fois que cela sera nécessaire : par exemple lorsqu'une nouvelle valeur de position aura été stockée dans l'attribut `Position`.

Du point de vue d'un outil d'interaction, la communication avec un objet interactif se déroule en trois étapes :

1. l'initialisation ;
2. l'utilisation de l'objet, c'est-à-dire l'envoi de données vers cet objet interactif afin de lui proposer de nouvelles valeurs ;
3. le relâchement du contrôle des attributs de cet objet interactif.

Cette séquence correspond à celle rencontrée lors d'une interaction où l'utilisateur sélectionne un objet, puis le manipule et enfin le relâche. Nous avons écrit qu'un outil d'interaction pouvait être manipulé par un utilisateur humain ou un système logiciel. Décrivons une interaction mettant en scène un humain.

Sélection. D'abord, l'utilisateur désigne un objet interactif qu'il souhaite manipuler. Par exemple, cette désignation peut s'effectuer par un dispositif logique piloté par un périphérique matériel. Puis, l'utilisateur sélectionne l'objet souhaité. La sélection est l'étape où l'utilisateur verrouille/confirme le choix de l'étape de désignation. Selon les techniques d'interaction, la désignation et la sélection peuvent être deux étapes confondues (ex : choix d'un objet interactif dans la liste des éléments d'un menu). Ces étapes pouvant conduire à une manipulation sont décrites par le diagramme d'activité présenté sur la figure 3.1.

Manipulation. Suite à la sélection, l'utilisateur peut à présent manipuler certains attributs d'un objet interactif. L'outil indique à l'utilisateur les modifications pouvant être apportées. Les modifications possibles sont connues de l'outil parce qu'il a préalablement interrogé l'objet interactif. Certains outils ne proposent pas à l'utilisateur les modifications possibles mais en essaient certaines d'emblée. Par exemple un outil qui a été créé dans le but de déplacer des objets essaiera systématiquement de modifier l'attribut `Position` d'un objet interactif.

Lorsqu'il manipule, l'outil passe en *état de manipulation*. L'outil envoie alors des valeurs à un objet interactif (ex : une nouvelle position, une nouvelle couleur, etc.).

[2] Ce type est souvent utilisé par des bibliothèques de rendu 3D et contient les données pour appliquer les opérations suivantes : une translation, une rotation et éventuellement un changement d'échelle.

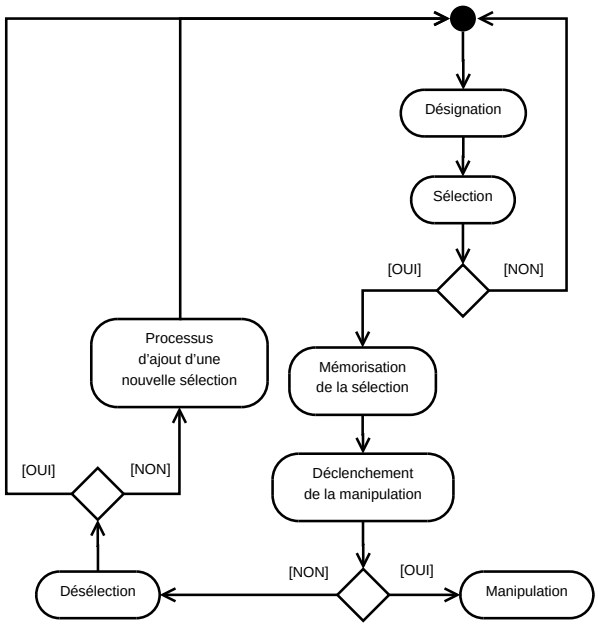


FIG. 3.1 – Diagramme d'activité de la sélection d'un objet. La désignation d'un objet est présentée comme une étape nécessaire à la sélection. La manipulation n'aura lieu que sur des objets sélectionnés. Enfin, des sélections multiples sont possibles.

Dans le cas où plusieurs outils manipulent simultanément le même objet, nous disons que ces outils agissent *coopérativement* et l'objet doit gérer par lui-même les valeurs multiples qu'il reçoit.

Relâchement. La phase de libération suit celle de manipulation. L'outil relâche son contrôle sur l'objet et ce dernier n'est plus manipulé.

3.4.2 Objets interactifs

Un objet interactif doit fournir aux outils la connaissance des modifications qu'il propose et autorise. Les objets interactifs sont construits par symétrie avec les outils d'interaction. Ainsi, un objet interactif est muni d'attributs qui représentent son état (ex : position, couleur).

Cependant, un outil d'interaction n'accède pas directement à un attribut mais passe par l'intermédiaire d'un *connecteur*. Prenons le cas de la position d'un objet interactif : elle peut être mise à jour par une nouvelle position absolue dans un repère, ou bien par une position relative à un repère local (ex : cas d'un objet qui est supporté par un autre). Par conséquent, un connecteur fournit différents moyens pour modifier un attribut. Un objet interactif, interrogé à propos de ses attributs, répondra par une liste de ses *connecteurs* ainsi que leurs niveaux d'accès.

Un mécanisme de politique d'accès est mis en place au niveau des connecteurs. Cette politique peut permettre :

1. d'empêcher des modifications provenant d'outils ayant un niveau d'accès faible. Une hiérarchie est ainsi introduite parmi les outils d'interaction. En fonction des propriétaires des outils d'interaction, il devient alors possible, par exemple, d'interdire l'accès du ballon à un joueur de football sur un terrain de sport ;
2. d'empêcher des modifications temporairement. Nous verrons que les outils peuvent modifier les permissions d'un objet interactif sous certaines conditions. Dans le cas d'un joueur de football, il est donc possible de lui interdire l'accès au ballon lorsque celui-ci est en possession de l'arbitre.

Tout attribut est typé. Du côté des outils d'interaction, chaque attribut est associé à une catégorie. Du côté des objets interactifs, chaque connecteur est associé à un seul attribut de l'objet interactif. De plus, chaque connecteur d'un objet interactif est associé à une catégorie et, pour savoir s'il peut être partagé, caractérisé par un niveau d'accès et un type d'accès. Le type de l'attribut et la catégorie sont utilisés pour faire correspondre un attribut d'un outil d'interaction à un connecteur d'un objet interactif.

Un connecteur partageable doit être capable de gérer des actions concurrentes provenant de plusieurs outils d'interaction. Dans ce but, un connecteur peut faire appel à un *convertisseur*. Un convertisseur construit une donnée à partir de celles envoyées par les outils d'interaction qui lui sont connectés. Par exemple, il peut calculer une moyenne pour plusieurs translations proposées. Les cas présentant des opérations non-commutatives sont difficiles à traiter, comme par exemple une séquence de rotations. Si l'ordre de traitement n'est pas celui effectué par les interacteurs, la rotation finale sera différente de celle souhaitée. Enfin, des valeurs discrètes peuvent être contradictoires et ne peuvent être gérées automatiquement sans une intervention de l'utilisateur. Par exemple, si deux outils d'interaction envoient des booléens simultanément, il sera difficile de faire un choix entre *vrai* et *faux* à la réception : il faudrait une politique de choix adaptée à la nature des attributs et au contexte d'utilisation.

3.4.3 Relations entre outils d'interaction et objets interactifs

La figure 3.2 montre la relation entre deux outils d'interaction et un objet interactif. Une partie de l'outil d'interaction est dédiée à la gestion de l'interaction. Cette partie est, par ailleurs, un composant logiciel réutilisable, que nous nommons *extension*, qui sera explicité dans ce chapitre (cf. paragraphe 3.6). Cette extension produit une nouvelle valeur pour l'objet interactif en cours de manipulation et la place dans un attribut de l'outil afin de l'envoyer. De façon à suivre l'évolution de l'objet interactif contrôlé, l'extension « écoute » également l'objet interactif (ex : suivi de position, arrivée d'un nouvel outil). Enfin, c'est aussi l'extension logicielle qui implémente notre protocole de communication.

Par symétrie avec un outil d'interaction, une extension rassemble la gestion de la communication avec des outils. Une partie de l'extension contient elle-même du code dont le rôle est de fusionner des commandes provenant de plusieurs outils en une seule commande : c'est le *convertisseur*.

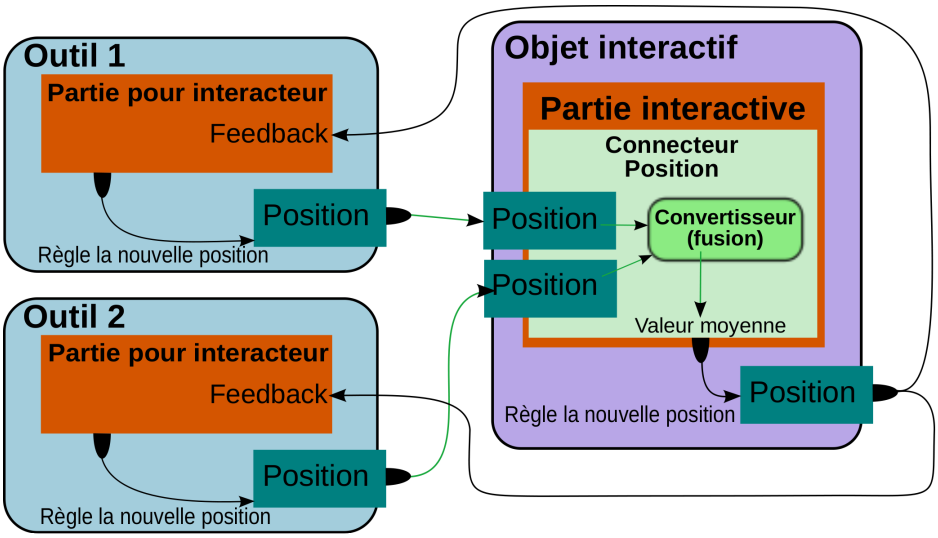


FIG. 3.2 – Illustration présentant deux outils proposant chacun une position à un objet interactif.

Actuellement, la correspondance entre un attribut d'un outil d'interaction et un connecteur d'un objet interactif s'effectue de façon rudimentaire. Une correspondance est trouvée par la paire formée de « catégorie » et « type de donnée ». En cas de multiples correspondances, seule la première paire sera employée.

Enfin, l'extension d'un outil d'interaction, tout comme celle d'un objet interactif, peut être dynamiquement ajoutée à un objet quelconque pour le transformer en un outil d'interaction ou en un objet interactif. L'opération peut aussi être renversée en supprimant l'extension.

3.5 Créer une communication entre outils d'interaction et objets interactifs

Nous expliquons d'abord comment un objet interactif peut être manipulé par plusieurs outils d'interaction. Puis, nous décrivons la façon dont un outil d'interaction peut manipuler un ou plusieurs objets interactifs simultanément.

3.5.1 Un objet interactif et plusieurs outils d'interaction

Nous considérons les deux façons de manipuler un objet interactif virtuel : soit un seul outil d'interaction manipule l'objet interactif, soit il y a au moins deux outils d'interaction en train de manipuler le même objet interactif en même temps.

3.5.1.1 Contrôle simple

La séquence que nous décrivons à présent est illustrée par la figure 3.3, elle montre comment un outil d'interaction prend le contrôle de tous les connecteurs d'un objet interactif qui ont été identifiés accessibles.

D'abord, l'outil ouvre une session avec l'objet qui a été précédemment sélectionné. L'ouverture de session est requise pour des raisons principalement techniques : des structures internes pour une communication peuvent nécessiter une initialisation. La sélection de l'objet a pu être effectuée par une entité logicielle ou une personne (*via* un menu par exemple).

À présent, l'outil d'interaction a besoin d'interroger l'objet interactif afin de connaître la liste des connecteurs qu'il pourrait manipuler :

1. il envoie la commande `get_accessible_parameters` à l'objet interactif ;
2. l'objet interactif répond par la commande `accessible_parameters` qui est composée de la liste des identifiants des connecteurs accessibles.

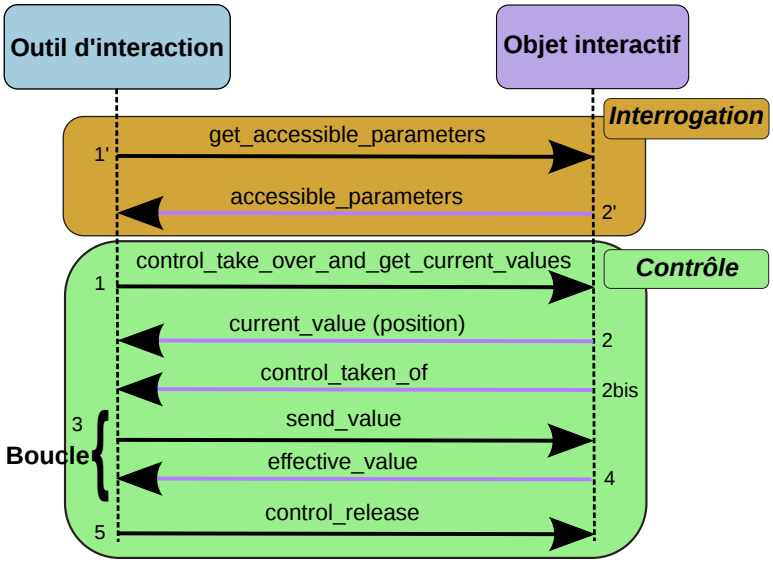


FIG. 3.3 – Séquence de communication entre un outil d'interaction et un objet interactif.

Maintenant, l'outil d'interaction peut essayer de prendre le contrôle de certains des connecteurs de l'objet interactif et, plus tard, de leur envoyer des valeurs :

1. l'outil essaie de prendre le contrôle des connecteurs accessibles. Par exemple, s'il essaie de contrôler un connecteur associé à l'attribut `Position`, il doit envoyer la commande `control_take_over_and_get_current_values` accompagnée de l'identifiant du connecteur considéré de `Position` ;
2. l'objet interactif utilise le message `current_value` pour renvoyer la valeur de la position. Ce message est composé de l'identifiant du connecteur employé et de la

valeur contenue dans l'attribut associé (ex : la position). Lorsque toutes les valeurs qui intéressent les outils d'interaction ont été envoyées, l'objet interactif envoie le message `control_taken_of` qui est composé des identifiants des connecteurs manipulés. Ce message agit donc aussi comme un message de fin de transmission. À partir des valeurs reçues, l'outil peut s'initialiser pour des calculs ultérieurs. Par exemple, avec une position, un outil d'interaction peut calculer un écart à maintenir entre lui-même et l'objet interactif ;

3. l'outil peut maintenant proposer de nouvelles valeurs à l'objet interactif avec la commande `send_value`. Elle contient l'identifiant de l'attribut et sa valeur associée. L'outil enverra autant de messages qu'il y a de valeurs à envoyer ;
4. l'objet interactif informe périodiquement les outils le contrôlant des valeurs des attributs par `effective_value`. Chaque message contient la valeur d'un attribut contrôlé, ainsi que son identifiant ;
5. l'outil peut relâcher le contrôle d'un connecteur par le message `control_release`. Ce message contient les identifiants des connecteurs dont le relâchement du contrôle est demandé.

3.5.1.2 Contrôle partagé

Un contrôle partagé est un contrôle où au moins deux outils d'interaction sont en train de manipuler un même objet interactif en même temps. Dans ce cas, chaque outil d'interaction a besoin :

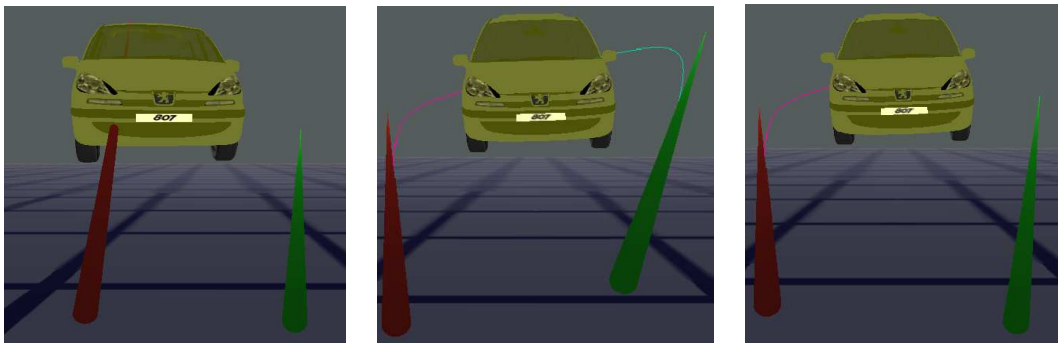
- de connaître la liste des autres outils en train de co-manipuler cet objet d'interaction ;
- de suivre l'évolution de l'objet interactif pour informer ses interacteurs (humains ou logiciels) que des outils d'interaction commencent avec lui une manipulation ou la terminent ;
- d'adapter son propre comportement à la situation de la session interactive.

Considérons un outil d'interaction noté T_1 qui est en train d'interagir avec un objet interactif noté O. Si un autre outil d'interaction, noté T_2, prend le contrôle de certains des connecteurs de O alors T_1 recevra un message `control_taken_by` composé de l'identifiant de T_2 et des identifiants des connecteurs contrôlés. Lorsque T_2 relâche le contrôle de certains connecteurs, T_1 recevra un message `control_released_by` composé d'informations de la même nature que pour `control_taken_by`.

La figure 3.4 illustre comment deux rayons manipulés par deux utilisateurs peuvent déplacer une voiture ensemble. Les rayons se courbent lorsque les positions proposées se contraignent. Cette métaphore est décrite dans [RHWF06] et [DF02].

Les accès concurrents sont gérés par l'objet interactif lui-même. Le message `control_take_over_and_get_current_values` n'est pas décomposé en deux commandes pour les raisons suivantes :

- si le contrôle est d'abord demandé (avec « control take over »), suivi de la demande des valeurs (avec « get current values »), alors un délai apparaît entre l'instant de prise de contrôle de l'objet interactif et sa manipulation. Or, pour éviter des



(a) (b) (c)

FIG. 3.4 – (a) Un seul rayon (le plus à gauche, rouge) contrôle la position de la voiture. (b) Les deux rayons sont courbés parce qu'ils proposent des valeurs de positions différentes pour la voiture. Une ligne courbée est une courbe de Bézier à trois points de contrôle dont le premier est l'origine du rayon, le deuxième est l'autre extrémité du rayon et le troisième correspond à la position de la voiture. (c) Le rayon à droite (le vert) ne contrôle plus la position, elle est uniquement manipulée par le rayon restant (celui à gauche, le rouge).

changements durant ce délai, il faudrait poser un verrou sur l'objet interactif lors de la demande de contrôle. Par conséquent, on empêcherait d'autres outils d'interaction de commencer une manipulation pendant cette période ;
– si les valeurs sont demandées avant la prise de contrôle, alors les valeurs utilisées lors de la prise de contrôle risquent d'être obsolètes. Cette situation est importante lorsqu'un utilisateur cherche à manipuler un objet interactif en mouvement.

3.5.2 Un outil d'interaction et plusieurs objets interactifs

Nous considérons qu'un outil d'interaction ne peut manipuler un objet interactif que si l'objet interactif a une extension interactive. Nous expliquerons comment un outil d'interaction interagit avec un objet interactif. Puis, nous expliquerons ce qu'est un objet interactif complexe et comment les outils d'interaction peuvent interagir avec.

3.5.2.1 Contrôle d'un seul objet interactif simple

Ce cas est celui qui a été présenté dans la partie 3.5.1. Un outil d'interaction sélectionne un objet interactif dans le but de manipuler ses attributs à travers des connecteurs. Puis, il interroge l'objet interactif pour connaître la liste des connecteurs présents ainsi que leurs propriétés. L'outil d'interaction essaie de prendre le contrôle de certains connecteurs. À la fin de la manipulation, l'outil d'interaction libère les connecteurs qu'il contrôlait.

3.5.2.2 Contrôle d'un seul objet interactif complexe

Nous qualifions un objet interactif de complexe lorsqu'il présente un très grand nombre de connecteurs pour interagir avec lui. Ce grand nombre requiert une approche particulière pour l'interrogation de l'objet interactif.

La construction de certains objets virtuels peut nécessiter une composition d'objets interactifs au lieu d'un seul objet interactif : une voiture virtuelle peut ainsi être faite de nombreux éléments (portes, roues, moteur, châssis, etc.), qui sont, chacun, contrôlés par une extension interactive. Certaines contraintes peuvent être ajoutées aux objets interactifs grâce à des extensions logicielles dans le but, par exemple, de maintenir les portes attachées à la voiture, de telle sorte que l'ensemble de la voiture puisse être déplacé tout en la conservant en un seul bloc.

Considérons un tuyau flexible. Dans le monde virtuel, nous proposons d'utiliser un ensemble de cylindres mis bout à bout pour le représenter. Pour conserver un bel aspect du flexible, il faudrait le discrétiser en un nombre élevé de cylindres et conserver ses cylindres parfaitement joints. De plus, pour des raisons de performances, on chercherait généralement à minimiser le nombre de cylindres employés. Contrairement au cas présenté avec une voiture, une approche globale s'avèrerait nécessaire. Un seul objet interactif, ou un nombre minimum d'objets interactifs, gèrerait alors l'ensemble de l'interaction avec le tuyau. Par exemple, la résolution d'équations d'un système physique par la méthode des éléments finis permettrait d'obtenir les positions et orientations des anneaux. Pour offrir des interactions (ex : tordre ou déplacer le tuyau) à un utilisateur, les anneaux devraient être rendus sélectionnables. Autrement dit, nous associerions chaque anneau à un attribut de position et l'utilisateur pourrait modifier l'une de ces positions en passant par un seul objet interactif pour tout le tuyau. Ces associations anneaux/attributs seraient décrites dans un fichier de configuration de l'environnement virtuel.

Un objet complexe doit donc être décomposé en éléments sélectionnables. Selon les bibliothèques logicielles responsable de la modélisation, ou du rendu, d'objets en trois dimensions, un maillage peut être décomposé en sous-maillages. Dans le moteur d'affichage Ogre3D [Ogr] qui a été employé pour implémenter les tests, cette notion est disponible. En cas d'absence, cette notion peut être aisément obtenue *via* des structures de données à implémenter. Chaque sous-maillage est associé à un identifiant qui est unique au sein de l'objet interactif virtuel auquel il appartient. L'association identifiant/sous-maillage est effectuée par le biais d'un fichier de configuration. Lorsqu'un utilisateur sélectionne une partie d'un objet, cette partie est un sous-maillage qui renverra donc son identifiant associé. Ensuite, l'outil virtuel peut alors questionner l'objet interactif pour connaître les connecteurs disponibles pour le sous-maillage sélectionné uniquement. La commande utilisée est `get_accessible_parameters` et comporte une paire d'identifiants : celui de l'objet interactif global accompagné de celui de la partie sélectionnée. L'identifiant d'une partie d'un objet sert donc de filtre sur les connecteurs d'un objet interactif.

Comme exemple simple, la figure 3.5 montre un widget 3D constitué de deux parties. À l'écran, il y a deux maillages 3D (une sorte de tube sur lequel coulisse un anneau)

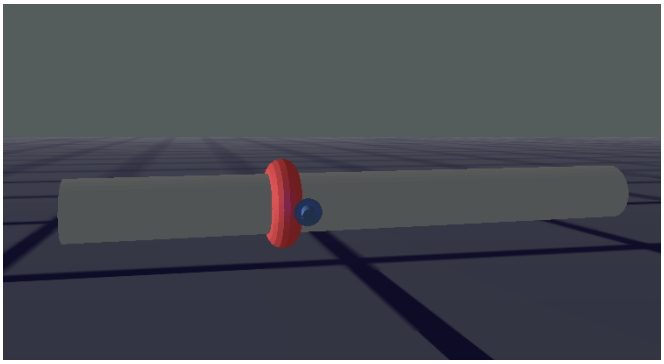


FIG. 3.5 – Exemple simple d'un objet interactif virtuel complexe constitué de deux parties : un tube et un anneau qui ne peut glisser que le long du tube.

mais un seul objet interactif est employé pour gérer des interactions avec l'ensemble de ce widget. Si un interacteur sélectionne l'anneau, il peut uniquement le déplacer le long du tube parce que l'objet interactif assure la préservation de deux contraintes : l'anneau doit glisser le long du tube et ne peut en être extrait. Si l'interacteur sélectionne le tube, c'est l'ensemble du widget qui sera déplacé.

3.5.2.3 Contrôle d'un ensemble d'objets interactifs

S'il essaie de manipuler simultanément un ensemble d'objets interactifs, un outil d'interaction n'interagit pas directement avec les objets interactifs mais à travers des *outils-proxys* (cf. figure 3.6).

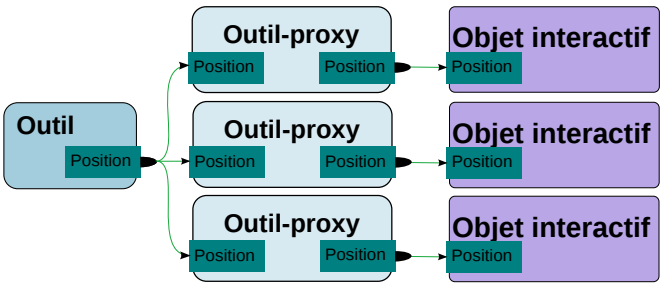


FIG. 3.6 – Un outil d'interaction communique ici avec trois objets interactifs grâce à trois outils-proxys.

Un outil-proxy est un outil d'interaction intermédiaire qui se place entre le « vrai » outil d'interaction et l'objet interactif manipulé. Pour simplifier, la présence des outils-proxys a été jusqu'à présent éludée de la description du protocole. Pourtant, un outil-proxy est systématiquement créé même pour interagir avec un seul objet interactif. Notons qu'il pourrait s'avérer nécessaire de ne produire des outils-proxys qu'en cas de

réelle nécessité, cependant l'implémentation s'en trouverait complexifiée. Par ailleurs, la copie inutile de données peut-être évitée grâce à un mécanisme de mémoire partagée si l'outil d'interaction et l'outil-proxy sont dans un même processus, ce problème incombe alors aux couches basses de communication.

L'intérêt des outils-proxys est d'ajouter plus de flexibilité dans la façon dont un outil est programmé : un outil n'a plus à contenir du code pour communiquer directement avec un objet interactif ou plusieurs. Un outil-proxy encapsule le code pour manipuler un objet interactif : une *extension de contrôle* et un *comportement d'outil d'interaction*. De cette façon, un outil d'interaction n'a plus qu'à instancier autant d'outils-proxys que d'objets interactifs à manipuler, puis suivre leurs évolutions.

En outre, un outil-proxy peut embarquer des extensions logicielles qui lui sont propres et différentes de celles de l'outil d'interaction parent. Ces extensions peuvent alors modifier la manière d'agir de l'outil-proxy auquel elles appartiennent. Enfin, la distribution d'outils-proxys sur des machines distantes peut également améliorer les performances en tirant profit de différences de charges sur des machines et de l'encombrement réseau.

3.5.3 Accès aux propriétés des objets interactifs

Les plates-formes de réalité virtuelle peuvent être employées pour simuler des interactions de la vie réelle où les différents protagonistes disposent de permissions restreintes pour leurs actions. Par exemple, un groupe d'interacteurs peut représenter un groupe d'utilisateurs ayant accès à des documents confidentiels dans un environnement virtuel. Il est également possible d'imaginer que certaines actions puissent devenir ponctuellement inaccessibles. Par exemple, lorsqu'un formateur forme des techniciens à des procédures de maintenance, il souhaite être le seul à réaliser une action pour la montrer, puis ce sera au tour des apprenants de reproduire l'action.

Pour implémenter un mécanisme d'accès aux connecteurs d'un objet interactif, nous proposons de manipuler des *propriétés interactives*. Chaque connecteur d'un objet interactif possède un niveau d'accès. Lorsqu'un outil d'interaction essaie de manipuler un connecteur, il a d'abord besoin de posséder un attribut qui correspond au connecteur souhaité de l'objet interactif : même type, méta-données correspondantes et un niveau d'accès suffisant. Avec cette politique, la prise de contrôle par un outil de certains connecteurs peut échouer, ou bien le contrôle de connecteurs peut être brutalement interrompu par un autre outil. En effet, lorsqu'un outil est en train d'interagir avec un objet interactif, un autre outil peut venir et augmenter le niveau d'accès requis. Ce changement conduit à une perte de contrôle pour le premier outil ainsi qu'à sa réception du message `control_ended` qui est composé des identifiants des connecteurs perdus.

3.5.3.1 Le contenu d'une propriété interactive

Les *propriétés interactives* peuvent être associées avec les objets interactifs ou leurs connecteurs. Décrivons les éléments d'un telle propriété :

- un **nom**. Par exemple, le nom pourrait être `Position` pour la position d'un objet interactif ;
- un **type**. Par exemple, un quaternion pour une rotation ou un vecteur pour une translation ;
- une **catégorie**. Par exemple, la position est exprimée dans un repère absolu ou bien local ;
- une **description**, c'est-à-dire une sorte de commentaire ;
- le **nombre d'outils qui peuvent interagir** avec l'objet interactif simultanément ;
- un **niveau de priorité** afin de déterminer la priorité minimale requise par un outil d'interaction pour avoir l'autorisation d'interagir avec l'objet interactif. Cette priorité peut être réglée globalement pour tous les objets interactifs, ou pour des groupes d'objets interactifs, ou individuellement pour un objet interactif. Elle peut également être réglée pour l'ensemble des propriétés d'un objet interactif ou individuellement pour chacun de ses connecteurs ;
- une **politique d'accès pour un connecteur** d'un objet interactif ou d'un groupe d'attributs. Elle détermine comment les interactions peuvent être partagées entre plusieurs outils d'interaction ;
- une **description** de la façon dont le retour à l'utilisateur employé pour le tenir au courant de ses actions (avec un attribut, un objet ou un groupe d'objets) doit être déclenché : avant, au début, pendant ou après la sélection et/ou la manipulation d'un objet interactif ou d'un attribut.

3.5.3.2 Politiques d'accès prédéfinies

Une politique d'accès pour un connecteur peut être créée par un utilisateur *via* la programmation de cette politique. Nous fournissons trois politiques par défaut.

Politique « unfreezable ». Un outil interagissant avec un tel connecteur ne pourra pas interdire un autre outil, autorisé, de stopper ou joindre l'interaction en cours. But : empêcher un outil déjà en interaction d'empêcher un outil de priorité suffisante soit de le rejoindre dans une manipulation coopérative, soit de l'exclure de la manipulation.

Politique « freezable at any level ». Un outil interagissant avec un tel connecteur sera autorisé à modifier, pour chaque groupe existant d'outils, le niveau minimum requis par un outil pour être capable de stopper ou joindre l'interaction actuelle. Ce niveau minimum pourrait aussi bien être fixé à une valeur extrêmement élevée afin de se protéger au maximum de toute intrusion durant une manipulation, que fixé à une valeur extrêmement faible pour permettre à des outils de priorité faible de se joindre à une manipulation collaborative. But : un outil pourra ainsi interdire l'accès à d'autres outils ou au contraire donner un accès à un groupe d'outils précis.

Politique « freezable at tool level ». Même politique que « freezable at any level » sauf que le niveau fixé ne peut être supérieur à la priorité de l'outil qui a initié l'in-

teraction : il ne peut donc pas se protéger d'outils ayant des priorités supérieures à la sienne.

3.6 Implémentation du protocole

Cette section décrit les différentes extensions logicielles que les outils d'interaction et les objets interactifs utilisent pour ajouter des fonctionnalités et implémenter notre protocole de communication.

3.6.1 Spécialisation par agrégation

Bien souvent, la spécialisation du comportement d'une classe dans un langage objet (en C++ dans notre cas) passe par un héritage. Cette méthode est souvent la première enseignée dans les cours de programmation orientée objet pour montrer aux étudiants comment ajouter des fonctionnalités à une classe ou les modifier. Malheureusement, l'héritage est source d'une grande complexité lorsque la hiérarchie qu'il fait apparaître comporte beaucoup de niveaux. Il devient délicat d'intégrer à l'héritage existant des nouvelles classes filles sans créer d'importantes duplications de code ou des liens/hérarchies qui n'ont pas lieu d'être d'un point de vue sémantique.

OpenMASK [LCAA08] est la plate-forme utilisée pour les implémentations décrites dans ce manuscrit. Elle utilise des *objets simulés* qui constituent la brique de base pour construire une application de réalité virtuelle. Un objet simulé est une classe contenant le code décrivant l'évolution d'entités constituants la simulation virtuelle. Pendant longtemps, le modèle de développement d'OpenMASK reposait sur l'héritage de la classe `SimulatedObject`. Dans les classes filles, l'ajout de fonctionnalités passait par l'usage d'une classe générique[3] paramétrée par la classe à modifier et la classe apportant un nouveau comportement. C'est de cette façon que des objets simulés étaient transformés en outils d'interaction, en objets interactifs, ou encore que leur fonctionnement était enrichi (cf. [MAC+02], [DLT04]).

Une autre approche consiste à agréger un ensemble de composants logiciels qui seront exécutés tour à tour. Par conséquent, nous partons d'une classe « vide » dont la seule capacité est d'agréger des composants et de les exécuter. C'est l'ajout, le remplacement ou la suppression de composants qui conduit à la création de nouveaux comportements. Ces composants sont des *extensions logicielles*.

Cette approche permet une programmation fine puisqu'il suffit de changer ces composants pour changer un comportement. Un niveau de réutilisabilité élevé est également obtenu parce que ces composants sont souvent utilisables avec différentes classes. Par ailleurs, l'interface de programmation d'une extension, montrée par la figure 3.7, est simple pour permettre des développements rapides. Deux méthodes peuvent être appelées : `preComputeParameters` et `postComputeParameters`. Leurs noms indiquent que l'une sera appelée avant la méthode `computeParameters`, l'autre après. La méthode

[3] En anglais, une « template ».

`computeParameters` est la méthode qui permet de faire évoluer d'un pas de temps un objet simulé OpenMASK.

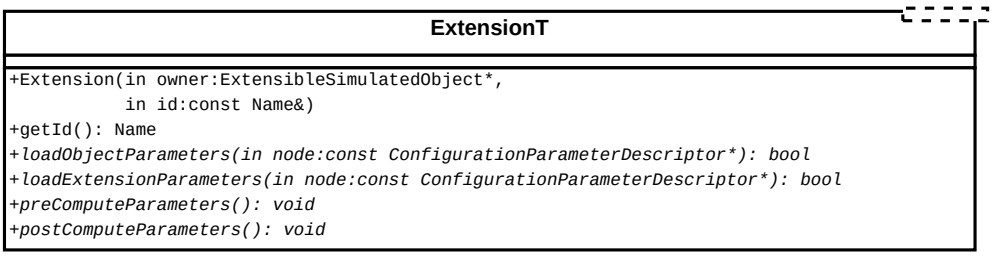


FIG. 3.7 – Diagramme de classes d'une extension paramétrable par un objet pour pouvoir accéder à certaines de ses propriétés. L'extension porte un identifiant, peut charger des configurations et offre deux méthodes pour effectuer des calculs.

Enfin, ces composants sont ajoutables ou supprimables dynamiquement au cours d'une exécution d'une simulation virtuelle, ce qui permet :

1. de modifier dynamiquement le comportement d'objets au cours de la simulation virtuelle ;
2. de pouvoir créer une interface graphique de programmation où l'utilisateur pourrait produire aisément de nouveaux comportements en venant piocher une extension parmi celles existantes ;
3. d'éviter des recompilations du code suite à des modifications. Cette étape était indispensable auparavant.

3.6.2 Éléments constituant un outil d'interaction

Nous voyons un outil d'interaction comme une agrégation d'au moins trois *extensions logicielles* : *sélection*, *manipulation* et *contrôle*. Toutefois, l'extension de contrôle se place dans les outils-proxys (cf. figure 3.8). Il existe donc une instance de l'extension de contrôle par outil-proxy.

3.6.2.1 Sélection

Cette extension enregistre les identifiants des objets interactifs que l'interacteur a sélectionné. La simplicité de cette tâche permet d'implémenter un grand nombre de techniques différentes de désignation/sélection d'objets. Par exemple, un utilisateur peut sélectionner par un rayon laser virtuel, un menu ou sa voix. En outre, cette méthode est suffisamment générique pour être indifféremment employée par un agent virtuel ou un humain.

Pour pouvoir être averti de la sélection d'un objet, une extension doit implémenter l'interface `ISelectedObjects` et s'abonner auprès de l'extension de sélection. Nous utilisons ici le patron de conception *Observateur* [GHJV94]. Cette interface contient

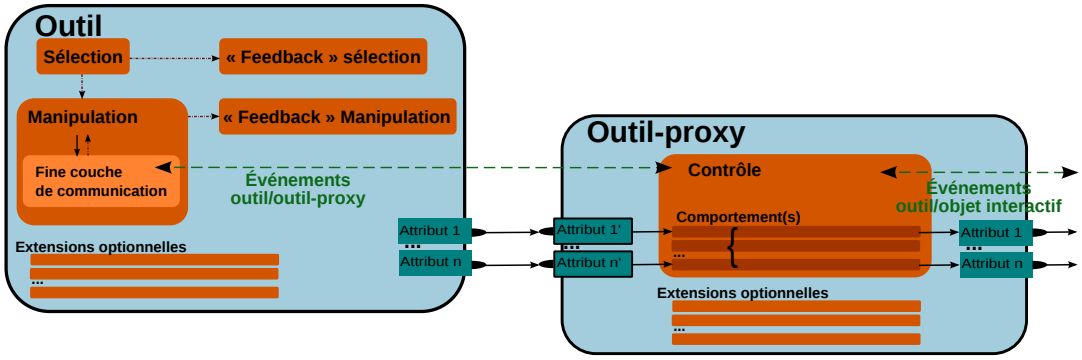


FIG. 3.8 – Les différentes parties constituant un outil d'interaction et un outil-proxy. L'outil et l'outil-proxy sont connectés ensembles. L'outil contient des extensions pour la sélection, la manipulation, ainsi que des extensions pour la perception de l'utilisateur qui seront explicitées plus tard. Enfin, une fine couche de communication, placée dans l'outil, dialogue avec l'extension de contrôle placée dans l'outil-proxy.

une méthode qui sera appelée à chaque fois que des objets auront été sélectionnés ou désélectionnés : elle prend l'ensemble des objets sélectionnés en argument. De la même façon, une extension souhaitant être informée de la désignation d'objets implémentera l'interface `IDesignatedObjects`. Cette interface est très similaire à `ISelectedObjects`.

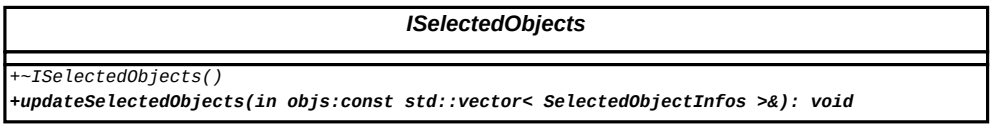


FIG. 3.9 – Interface à implémenter pour écouter des sélections.

3.6.2.2 Manipulation

Cette extension écoute des événements entrants pour ordonner des actions à l'*extension de contrôle*. L'*extension de manipulation* a également pour but d'implémenter la façon dont les outils réagissent lorsqu'ils reçoivent une information de l'extension de contrôle. Par exemple, l'extension de manipulation peut afficher un menu 3D à l'utilisateur pour qu'il choisisse un type d'action à effectuer. Ce menu apparaîtrait lorsque l'utilisateur veut interagir avec un objet interactif.

Cette extension est également responsable du routage de messages vers d'autres extensions dédiées à la prise de conscience. Il s'agit ici de donner à l'utilisateur un retour d'informations concernant ce qui se passe. L'extension de manipulation peut donc prendre la décision de ne pas router certains messages. Par exemple, on peut imaginer qu'un « feedback » utilisateur passant par l'affichage de menus ne nécessitera pas les mêmes informations que celui passant par un retour sonore.

Un humain peut démarrer une interaction en pressant un bouton d'un périphérique matériel, cela enverra un événement à l'extension de Manipulation qui l'interprétera. Cette extension logicielle essaie d'interagir avec les objets interactifs précédemment sélectionnés. Donc, elle implémente l'interface `ISelectedObjects` et s'abonne auprès de l'extension de sélection pour être capable d'écouter ses événements. La figure 3.10 montre un diagramme de classes UML où l'extension de manipulation hérite d'une interface `ISelectedObjects` pour avoir connaissance des objets sélectionnés. L'interface `IObjectControl` permet de suivre le déroulement de la manipulation, c'est-à-dire du contrôle de l'objet manipulé.

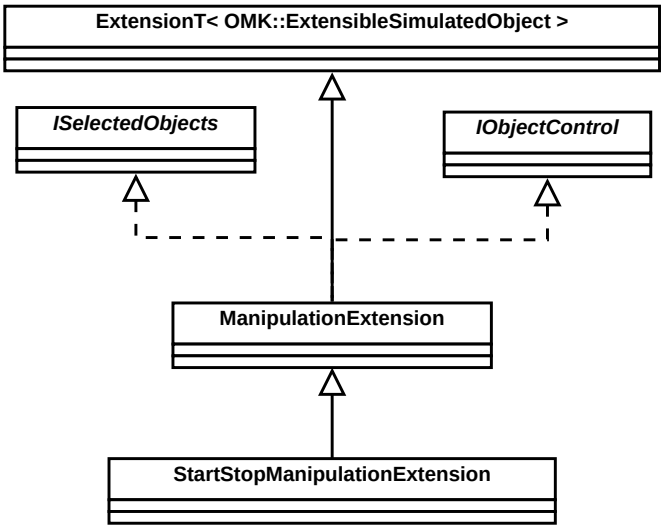


FIG. 3.10 – Diagramme de classes pour l'extension de manipulation qui ne permet qu'un démarrage ou une fin de manipulation.

Des automates à états finis sont utilisés pour décrire l'évolution de la communication entre un outil d'interaction et un objet interactif. Un automate est créé à chaque fois qu'un outil d'interaction essaie de prendre le contrôle d'un objet interactif. De plus, l'extension de manipulation utilise une machine à états par objet interactif que l'outil manipule. S'il y a plusieurs objets interactifs manipulés, elle fait évoluer les automates associés indépendamment. Les états de l'automate servent à envoyer des commandes (ex : `get_accessible_parameters`) tandis que les transitions entre deux états ne sont déclenchées que lorsqu'un événement attendu a été reçu. L'extension nommée `StartStopManipulationExtension` (cf. figure 3.10) fonctionne de la façon suivante : elle lance la manipulation de tous les objets sélectionnés, en créant un automate par objet à manipuler, et la manipulation d'un objet interactif s'arrête immédiatement lorsque l'envoi d'une commande entraîne une erreur (ex : l'objet interactif n'est plus joignable). Les objets interactifs manipulés sont libérés dès que l'extension reçoit un événement lui demandant d'interrompre ses manipulations.

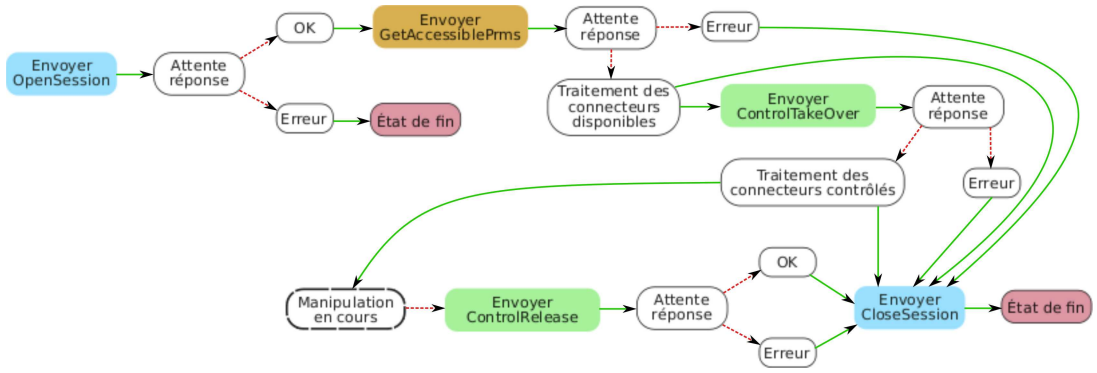


FIG. 3.11 – Automate gérant la manipulation d'un objet interactif. La plupart des échecs conduisent à un abandon complet de la tentative de manipulation. La manipulation a lieu si au moins un connecteur peut être manipulé.

La figure 3.11 présente l'automate employé par `StartStopManipulationExtension`. Sur celui-ci, l'outil commence par ouvrir une session avec l'objet interactif en envoyant la commande `OpenSession`, puis, il passe automatiquement en état d'attente d'une réponse. Si la réponse indique une erreur, l'outil abandonne l'ouverture de session et arrive dans un état de fin pour cet automate. Si la réponse est positive, l'automate enchaîne automatiquement avec une interrogation de l'objet interactif pour obtenir la liste de ses connecteurs. Il est donc à noter deux types de transitions : celles automatiques et celles qui n'opèrent que lorsque des conditions sont réunies.

Chaque état d'un automate peut être relié à plusieurs autres états via une *transition*. Une transition est un passage possible entre deux états. Le passage dépend d'un *prédicat*. L'automate suivra la première transition trouvée dont le prédicat est *vrai*. La fréquence de tentative de changement d'état correspond à la fréquence de fonctionnement de l'outil d'interaction.

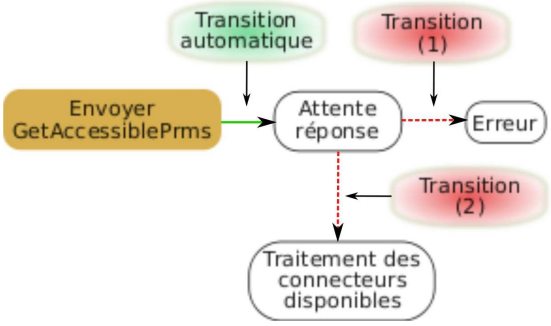


FIG. 3.12 – Détails d'une partie de l'automate de manipulation pour mettre en valeur les transitions. Ici, une transition automatique et deux transitions, (1) et (2), qui dépendent de conditions extérieures à l'automate.

La figure 3.12 détaille le moment où l'automate interroge un objet interactif afin de connaître ses connecteurs. Entre l'état d'envoi de la requête de demande et celui d'attente de réponse, le passage est automatique, autrement dit, un prédicat renvoyant toujours vrai est employé. Pour les transitions (1) et (2), le prédicat est conditionné par la réception du message `accessible_parameters` provenant de l'objet interactif. Ces deux transitions sont des objets C++ implémentant chacun l'interface `IObjectControl` pour suivre l'évolution du contrôle de l'objet interactif. Il suffit alors à ces transitions de vérifier que l'objet interactif répondant est celui que l'automate manipule. Tout comme ces transitions, l'état de traitement de la réponse va également s'abonner à l'extension de manipulation pour connaître les connecteurs disponibles. Une fois la transition (2) franchie, il devra vérifier que des connecteurs sont disponibles.

La figure 3.11 représente l'automate dans l'état « manipulation en cours » (*i.e.* l'outil lui envoie des valeurs). La sortie de cet état dépend d'une transition qui écoute un événement d'arrêt. Dans ce but, cette transition implémente une interface `IStopManipulation`. Par la suite, l'automate libère les connecteurs qu'il contrôle et clôt la connexion avec l'objet interactif.

3.6.2.3 Contrôle

Cette extension implémente la partie outil du protocole de communication et est présente dans chaque outil-proxy. Elle communique avec l'extension de manipulation de l'outil d'interaction grâce à une fine couche de communication. Cette fine couche est présente dans l'extension de manipulation de l'outil d'interaction. Elle est qualifiée de « fine » parce que ses fonctionnalités sont intentionnellement limitées et se basent sur un code basique : le code route les messages vers les outils-proxys (pour demander à un objet interactif ses connecteurs, pour essayer de prendre le contrôle, etc.) ou route ceux en provenance des outils-proxys vers l'extension de manipulation elle-même.

L'extension de contrôle est capable d'effectuer quatre types d'actions :

- ouvrir/fermer une session avec un objet interactif ;
- prendre/relâcher le contrôle de connecteurs d'un objet interactif ;
- suivre l'évolution du contrôle d'autres outils : un autre outil prend le contrôle, ou relâche le contrôle, de connecteurs que l'outil actuel est en train de manipuler.

Chacune des actions repose sur l'envoi d'un message à l'objet interactif, ou bien sur la réception d'un message provenant de ce dernier. Les messages reçus sont immédiatement remontés à l'extension de manipulation et permettent notamment de faire progresser l'automate et informer l'interacteur.

Comportements. L'extension de contrôle utilise également des objets C++ qui sont plus simples encore que des extensions pour décrire des comportements. Chacun de ces objets est un composant logiciel facilement interchangeable, agrégé à l'extension de contrôle. Un tel composant logiciel est associé à un connecteur. Lorsque, par exemple, le connecteur `Position` d'un objet interactif est manipulé, c'est le comportement associé au connecteur `Position` qui est exécuté. L'interface de programmation d'un comportement est constituée de trois méthodes :

- `init`, pour effectuer des calculs préparatoires à l'évolution du comportement ;
- `compute`, pour calculer une nouvelle valeur et la fournir à l'attribut associé ;
- `finish`, pour stopper les calculs et éventuellement libérer des ressources allouées pour les calculs.

3.6.2.4 Prise de conscience des interactions

Nous associons également à l'extension de sélection des extensions de prise de conscience pour les interacteurs (humains ou agents virtuels), afin de les avertir de la sélection/désélection d'objets interactifs[4].

Pour améliorer sa ré-utilisabilité, une extension pour la prise de conscience doit être décorrélée au maximum d'un mécanisme particulier de retour à l'utilisateur. Il est ainsi souhaitable que la même extension, avec les mêmes informations envoyées, puisse informer : la partie graphique, la partie sonore et éventuellement la partie haptique (ex : en utilisant des vibrations).

Une implémentation d'une extension de prise de conscience est réalisée avec l'extension `SendSelectionAwarenessExtension`. Son rôle est d'envoyer les chaînes de caractères « ON » ou « OFF » aux objets[5] capables de les interpréter. Ainsi, si nous souhaitons mettre en surbrillance un objet 3D lors de la réception de « ON », il faut alors rendre le système de visualisation capable d'écouter ces événements. Nous pouvons également appliquer cette méthode pour générer un son lors d'une sélection ou d'une désélection.

Le diagramme de classes présenté par la figure 3.13 montre que l'extension `SendSelectionAwarenessExtension` implémente l'interface `IDesignatedObjects`, pour suivre les désignations, et `ISelectedObjects`, pour suivre les sélections.

De la même façon, nous créons des extensions de prise de conscience pour la manipulation. Elles écoutent l'extension de manipulation pour connaître les connecteurs nouvellement contrôlés ou relâchés. Une extension de ce type peut, par exemple, être utilisée pour mettre en surbrillance un objet qui est en train d'être manipulé ou afficher des informations à destination des autres interacteurs du monde virtuel afin de faire percevoir qui manipule.

3.6.3 Éléments constituant un objet interactif

Un objet interactif a pour but d'être manipulé par un outil d'interaction. Par conséquent, les objets interactifs partagent des aspects communs avec les outils d'interaction. Cependant, un objet interactif a un rôle supplémentaire fondamental par rapport à un

[4]Nous réalisons volontairement une confusion entre objets interactifs et objets virtuels. Les utilisateurs ne verront que des objets virtuels. Un objet interactif peut être représenté par plusieurs objets virtuels. Comme les utilisateurs sélectionneront un objet virtuel, ce qui entrainera la sélection de l'objet interactif associé, il n'est pas faux d'écrire, par raccourci, que les utilisateurs sélectionnent des objets interactifs.

[5]Ici, le terme « objet » est pris au sens très large. Par exemple, on peut considérer tout objet C++ ou des objets de simulation d'OpenMASK.

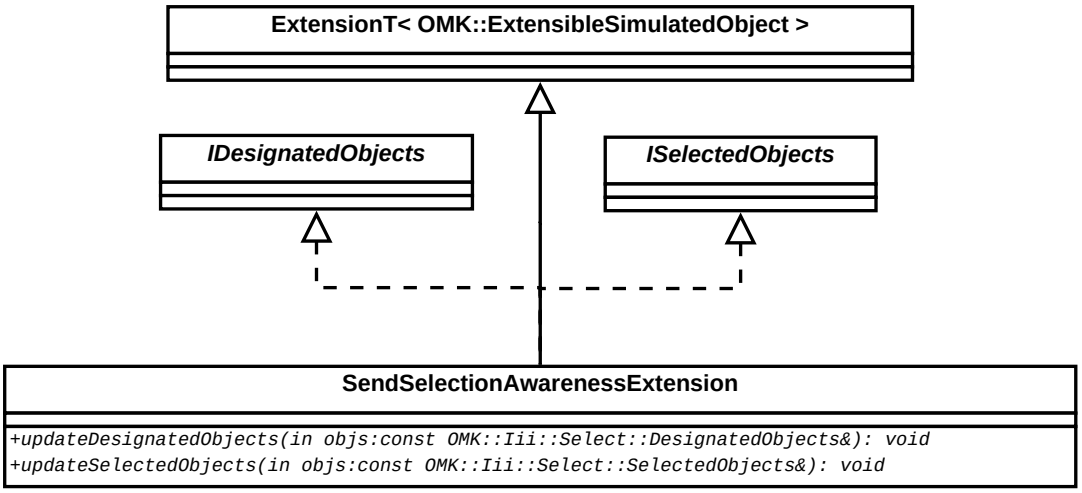


FIG. 3.13 – Diagramme de classes pour une extension qui informe des désignations et des sélections.

outil d'interaction. Pour un même attribut, il doit gérer les valeurs multiples qui lui sont envoyées par plusieurs outils d'interaction.

3.6.3.1 Symétrie avec les outils d'interaction

Les objets interactifs sont composés d'agrégats d'extensions logicielles. *A minima*, l'*extension interactive* doit être présente parce qu'elle implémente la partie objet-interactif de notre protocole de communication. L'extension doit répondre aux interrogations de l'outil d'interaction : connecteurs accessibles, demande de prise de contrôle de connecteurs, etc.

À la manière des extensions de prise de conscience des outils d'interaction, il est possible d'employer des extensions dans des objets interactifs pour suivre l'évolution d'une interaction. L'une des extensions permet notamment d'être informé de la prise de contrôle d'un connecteur ou de son relâchement. Il est également possible de placer ces mécanismes d'écoute au niveau des connecteurs pour profiter d'informations plus précises. L'intérêt de ce type d'extensions est de faire comprendre aux interacteurs ce qui est en train d'arriver à un objet interactif : qui le manipule et comment.

3.6.3.2 Fusion de valeurs envoyées par des outils d'interaction

L'objet interactif est responsable de l'interprétation des valeurs qui lui sont envoyées par des outils d'interaction afin d'en tirer des valeurs à stocker dans ses attributs.

Afin de combiner des valeurs provenant de plusieurs outils d'interaction qui interagissent avec lui, l'objet interactif dispose d'un *convertisseur* qui peut fusionner plusieurs valeurs en une seule. Nous avons cité le convertisseur qui calcule des moyennes

à partir de ses entrées. D'autres convertisseurs pourraient ajouter du bruit aux valeurs données, appliquer un facteur d'échelle, exclure certaines valeurs, etc.

D'un point de vue formel, un convertisseur projette un espace à n dimensions sur un espace à une seule dimension avec, pour les deux espaces, des valeurs qui doivent avoir le type de l'attribut associé au convertisseur. Cette situation suppose que n valeurs sont disponibles juste avant la fusion. Dans OpenMASK, une valeur manquante peut être obtenue, de façon transparente, par interpolation des valeurs précédemment reçues.

3.7 Description d'outils d'interaction et d'objets interactifs

Nous proposons des extensions au format COLLADA [COL] pour décrire les propriétés interactives d'objets interactifs. Ces travaux ne formalisent pas, pour l'instant, la configuration d'un outil d'interaction. Un outil d'interaction est donc décrit, certes par des balises XML, mais qui sont spécifiques à OpenMASK et, pour cette raison, nous n'en parlerons pas.

Il est à noter que la présentation fournie dans ce manuscrit cherche à donner une vision d'ensemble au lecteur. Les nombreux détails font partie de la documentation produite lors du travail sur le lot 1.2 de Part@ge qui vise à définir un format pivot d'échange de données entre plates-formes de réalité virtuelle et outils de création de contenus.

3.7.1 Utiliser le format OpenMASK ou COLLADA

Les outils d'interaction et les objets interactifs peuvent être décrits par deux moyens : le format historique d'OpenMASK et le format COLLADA. La raison de la conservation du format historique tient à la possibilité de pouvoir utiliser ces fichiers sur des plates-formes OpenMASK n'utilisant pas COLLADA, parce que, par exemple, la compilation du système pour COLLADA n'est pas jugée utile sur des projets menés de longue date qui sont déjà volumineux et complexes. En pratique, le développement a souvent été effectué sur le format d'OpenMASK puis a été porté sur COLLADA. Le travail de conversion est automatisable pour deux raisons. D'une part, le format OpenMASK présente, en fait, un grand ensemble de paires « un symbole / une valeur associée ». D'autre part, ces deux formats structurent tous deux l'information de façon hiérarchique et, donc, il s'agit principalement de réécrire une information destinée à OpenMASK en l'entourant de balises XML et de respecter les différences de hiérarchies qui peuvent exister entre les deux formats.

3.7.2 Une introduction au format COLLADA

Les fichiers COLLADA sont écrits en XML pour aider leur lecture et leur écriture par des logiciels de création de contenu multimédia (principalement pour des scènes 3D). Le format COLLADA se situe comme un format d'échange, un format pivot, entre un logiciel de modélisation 3D et un logiciel d'ajouts d'effets visuels, ou encore entre un logiciel

de modélisation et un logiciel ajoutant des propriétés physiques aux objets peuplant le monde en train d'être créé.

Dans le but d'être facilement extensible, le format COLLADA se divise en plusieurs parties : le noyau[6] COLLADA, la physique, la cinématique, les effets visuels. La partie pour la cinématique a ainsi vu le jour avec la version 1.5 de COLLADA.

Le noyau de COLLADA peut accueillir plusieurs scènes différentes : visuelle, physique, cinématique, effets visuels, qui peuvent être divisées en plusieurs modules appelés, chacun, bibliothèque[7], pour décrire plus aisément une scène complexe. L'exemple présenté sur la figure 3.14 montre une bibliothèque de scènes visuelles qui en comporte deux. Toute scène visuelle est composée d'un *nœud* fils comportant des éléments à afficher.

COLLADA emploie une structure arborescente avec à sa base les bibliothèques, elles-mêmes divisées en plusieurs scènes. Les bibliothèques contenues dans le noyau de COLLADA peuvent servir à la description d'animations, de caméras, de géométries, de lumières ou de scènes visuelles. Une scène peut comporter un nœud contenant lui-même plusieurs sous-nœuds. COLLADA présente ainsi une multitude de graphes de scène.

Pour éviter des redondances, tout nœud peut faire appel à plusieurs instances différentes définies en fonction de la partie de COLLADA à laquelle elles sont rattachées. Dans le noyau de COLLADA, on peut trouver : `instance_visual_scene` pour instancier une scène visuelle, `instance_camera` pour instancier la description d'une caméra, `instance_geometry` pour instancier un objet géométrique, `instance_node` pour instancier un nœud, etc.

Enfin, il nous faut préciser l'usage de la balise `extra` qui a été essentiel pour ajouter des informations nécessaires aux interactions collaboratives envisagées dans Part@ge. Cette balise permet d'étendre les fonctionnalités de la balise à laquelle elle se rattache. Avec la figure 3.14, elle se retrouve à étendre une bibliothèque de scènes visuelles. La balise `technique` utilisée à l'intérieur de `extra` permet de n'interpréter que le code qui est associé à un profil. Par rapport à la figure 3.14, un logiciel viendrait utiliser uniquement ce qui correspond au profil `profil_particulier`. Si un profil recherché n'est pas trouvé alors ce qui est associé à `technique_common` est employé par défaut. Donc, un logiciel ne comprenant pas les données rajoutées pour la collaboration dans Part@ge ne les chargera pas, ce qui permet de charger un tel fichier sur n'importe quel logiciel utilisant COLLADA sans perturber son fonctionnement.

3.7.3 Extension du format COLLADA pour la collaboration

Un nouveau profil, nommé `PARTAGE`, a été créé. Les ajouts au format COLLADA portent sur la description des propriétés interactives d'objets interactifs (cf. paragraphe 3.5.3.1). Il a été ajouté deux nouvelles bibliothèques :

- `library_interactive_scenes` qui contient des balises `interactive_scene` dont une instance est `instance_interactive_scene`. Une scène interactive peut comporter plusieurs `interactive_model` ;

[6] En anglais, « core ».
[7] En anglais, « library ».

```
<library_visual_scenes>
  <visual_scene id="VisualSceneNode" name="untitled">
    <node id="mon_objet" name="mon_objet">
      <translate sid="translation">0 0 0</translate>
      <instance_geometry url="#mon_objet_geo">
        ...
      </instance_geometry>
      <node id="une_sous_partie_mon_objet"
            name="une_sous_partie_mon_objet">
        <rotate sid="rotation_z">0 0 1 0</rotate>
        <instance_geometry url="#une_sous_partie_mon_objet_geo">
          ...
        </instance_geometry>
      </node>
      ...
    </node>
  </visual_scene>
  <visual_scene ...>
    ...
  </visual_scene>
  <extra>
    <technique_common>
      ...
    </technique_common>
    <technique profile="profil_particulier">
      ...
    </technique>
    ...
  </extra>
</library_visual_scenes>
<library_geometries>
  <geometry id="mon_objet" name="mon_objet">
    <mesh>
      ...
    </mesh>
  </geometry>
</library_geometries>
...
```

FIG. 3.14 – Exemple d'une description en COLLADA [COL] d'une bibliothèque visuelle comportant deux scènes visuelles. La première scène visuelle comporte un nœud utilisant une géométrie et comportant lui-même un sous-nœud. Les instances des géométries sont effectuées à partir des descriptions contenues plus bas.

– `library_interactive_models` qui contient des balises `interactive_model` dont une instance est `instance_interactive_model`.

L'organisation d'une bibliothèque de modèles interactifs, n'ayant qu'un seul modèle interactif, est donnée par la figure 3.15.

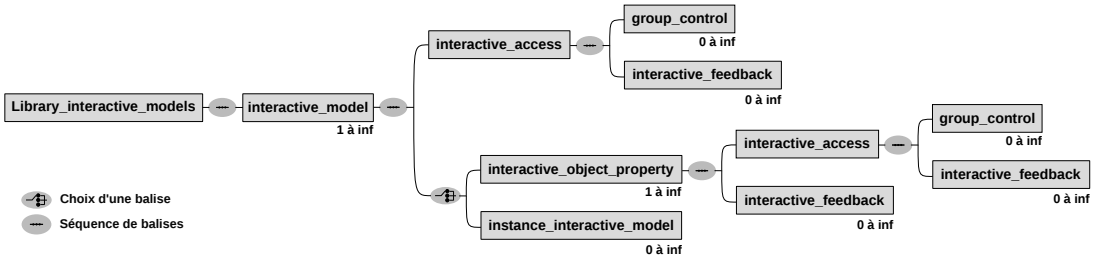


FIG. 3.15 – Hiérarchie des balises COLLADA pour l'interaction dans Part@ge.

Un exemple succinct de description en COLLADA d'un objet interactif est donné par la figure 3.16.

```
<partage:interactive_model id="mon_objet" target="mon_objet">
  <partage:interactive_access mode="ABSOLUTE" count="0"
                              policy="Freezable at tool level">
    <partage:group_control group="local" level="2" />
    <partage:group_control group="invite" level="1" />
    <partage:group_control group="inspecteur" level="3" />
    <partage:group_control group="*" level="0" />
  </partage:interactive_access>
  <partage:interactive_object_property
    name="mon_objet_Interact_TranslationXYZ"
    type="translation" type_meaning="translation_XYZ"
    target="mon_objet/translation" />
</partage:interactive_model>
```

FIG. 3.16 – Exemple succinct de description en COLLADA d'interactions définies pour un objet. Plusieurs groupes de niveaux différents sont définis pour les outils : local, invité, inspecteur, inconnu. Il est ici défini l'accès à l'attribut gérant le déplacement en translation de l'objet dont l'identifiant est « mon_objet ».

Balise interactive_access. Elle définit les conditions d'accès à un attribut. Les éléments de cette balise sont donnés par la tableau 3.1. Elle fait usage du type `AccessModeType` qui contient une chaîne de caractères : « ABSOLUTE » ou « RELATIVE ». Pour l'instant, cet attribut (au sens XML) prend du sens avec une position, c'est-à-dire si la position fournie est dans un repère absolu ou relatif. L'attribut `policy` est du type

`AccessPolicyType`, qui est une chaîne de caractères, pouvant valoir « Freezable at any level », « Freezable at tool level » ou « Unfreezable ». Ces valeurs correspondent aux politiques annoncées au paragraphe 3.5.3.2.

Élément interactive_access	
Séquence d'éléments	• `group_control`. Type : chaîne de caractères. Répétitions : 0 à ∞. Attributs XML : ↪ Nom : `group`. Type : chaîne de caractères. Défaut : « * ». ↪ Nom : `level`. Type : entier. Défaut : 0. • `interactive_feedback`. Répétitions : 0 à ∞.
Attributs XML	↪ Nom : `mode`. Type : `AccessModeType`. Default : ABSOLUTE. ↪ Nom : `count`. Type : entier. Défaut : 0. ↪ Nom : `policy`. Type : `AccessPolicyType`. Default : Unfreezable.

TAB. 3.1 – Description du contenu de la balise interactive_access. Le symbole « ↪ » indique un attribut XML. Tout attribut est associé à un élément XML.

Balise group_control. Elle définit le niveau requis par un outil pour accéder aux attributs de l'objet interactif en fonction de son appartenance à un groupe. Elle contient les propriétés suivantes :

- `group` : une chaîne de caractères représentant le nom du groupe auquel l'outil doit appartenir pour obtenir certains droits d'accès. Il est possible d'utiliser des caractères de substitution pour faire correspondre différentes chaînes de caractères à un même groupe. Le caractère `*` remplace un nombre quelconque de caractères et `?` en remplace un seul ;
- `level` : un entier représentant la priorité minimale requise par un outil.

Balise interactive_object_property. Elle définit les propriétés du modèle interactif, en particulier cette balise utilise des propriétés d'accès. Les éléments de cette balise sont donnés dans le tableau 3.2.

Balise interactive_feedback. Elle permet de mettre en place un mécanisme de retours d'informations, généralement vers l'utilisateur mais aussi, éventuellement, vers une entité logicielle. Elle comporte un attribut `stage` de type `FeedbackStageType` dont la valeur par défaut est AFTER. Cette balise indique le moment de déclenchement d'un

Élément interactive_object_property	
Séquence d'éléments	• `interactive_access`. • `interactive_feedback`.
Attributs XML	↪ Nom : `name`. Type : chaîne de caractères. ↪ Nom : `type`. Non optionnel. Type : `InteractivePropertyType`. ↪ Nom : `description`. Type : chaîne de caractères. ↪ Nom : `target`. Non optionnel. Type : chaîne de caractères.

TAB. 3.2 – Description du contenu de la balise interactive_object_property. Le symbole « ↪ » indique un attribut XML. Tout attribut est associé à un élément XML.

retour à l'utilisateur : « BEFORE » (avant la manipulation, c'est-à-dire pendant la phase de désignation de l'objet), « START » (au démarrage de la manipulation), « DURING », « AFTER » (dès que la manipulation est terminée, ce retour est généralement une transition vers l'état normal de l'objet).

3.8 Quelques exemples d'outils d'interaction implémentés

Afin de valider le modèle d'architecture décrit, différents outils d'interaction ont été implémentés.

3.8.1 Pointeur 3D

Un pointeur 3D est une extension du curseur d'une souris 2D classique, il s'agit ici d'un outil d'interaction dont la représentation graphique peut être déplacée dans les trois dimensions de l'espace virtuel. En outre, nous avons également ajouté la possibilité de l'orienter afin qu'il puisse appliquer cette transformation à l'objet interactif manipulé.

Un pointeur s'emploie comme suit : l'utilisateur déplace et oriente le pointeur pour pointer l'objet virtuel à manipuler, ce qui produit la désignation de l'objet. L'utilisateur verrouille la sélection (en appuyant sur une touche d'un clavier, en procédant par la reconnaissance vocale d'un mot, etc.). Durant cette étape de sélection, il peut aussi aller désigner et sélectionner d'autres objets virtuels. Enfin, il déclenche la manipulation des objets virtuels sélectionnés, c'est-à-dire qu'il commence à les déplacer. Dès que le pointeur est déplacé, tous les objets virtuels manipulés sont déplacés d'autant afin de maintenir un écart constant entre le pointeur et eux. Une séquence de ce type est illustrée par la figure 3.17, elle met en valeur l'utilisation de couleurs pour renseigner l'utilisateur sur les évolutions de la manipulation. Cependant, cette technique n'est

valable que pour des objets virtuels dont la couleur initiale peut être altérée d'une façon visible pour l'utilisateur et sans trop en dégrader l'aspect. De plus, le code couleur doit être maintenu à un faible nombre d'items pour permettre à l'utilisateur de les mémoriser.

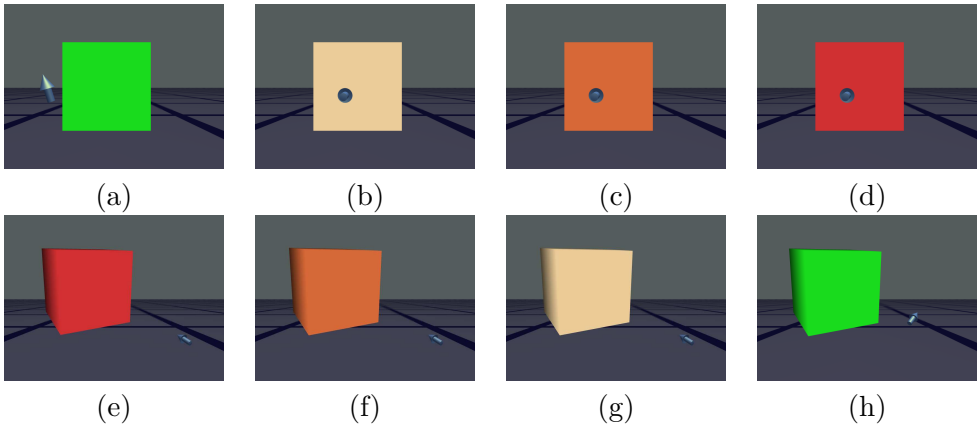


FIG. 3.17 – Les captures d'écran montrent les étapes de la manipulation d'un objet virtuel. (a) Le pointeur 3D (*i.e.* la flèche) ne pointe vers aucun objet. (b) Le pointeur désigne le cube. Pour montrer sa désignation, le cube se colore en beige. (c) Le cube a été sélectionné par l'utilisateur lorsqu'il a appuyé sur la touche « S » de son clavier. Le cube s'est coloré en orange. (d) L'utilisateur est en train de manipuler le cube. Ce dernier s'est coloré en rouge. (e) En le manipulant, l'utilisateur a déplacé le cube vers la gauche de l'écran. (f) L'utilisateur vient d'arrêter la manipulation du cube. Le cube reprend la couleur de sélection. (g) Le cube a été désélectionné. (h) Le cube n'est plus désigné.

Pour aider l'utilisateur à mieux savoir quel objet virtuel il est en train de manipuler, ou encore pour l'aider à repérer plus rapidement cet objet dans la scène, une ligne peut être tirée entre le pointeur et l'objet (cf. figure 3.18). Cette aide s'avère tout particulièrement utile lorsque l'utilisateur est en train de manipuler plusieurs objets simultanément. Sur la figure 3.19, l'utilisateur a désigné le parallélépipède vertical et l'a sélectionné, puis il a désigné le cube et l'a sélectionné, enfin il est en train de manipuler les deux objets à la fois.

3.8.2 Rayons droits et coudés

Un rayon est un outil d'interaction permettant de sélectionner des objets virtuels en les touchant par le bout du rayon, voire en les « embrochant ». Il permet également à l'utilisateur de déplacer des objets virtuels et de les orienter. La figure 3.20 montre un rayon en train de manipuler un cube. Ce rayon pourrait être lui-même manipulé par un pointeur 3D, ce qui montre l'aspect générique des outils d'interaction en les chaînant. Un rayon peut donc être à la fois outil d'interaction et objet interactif.

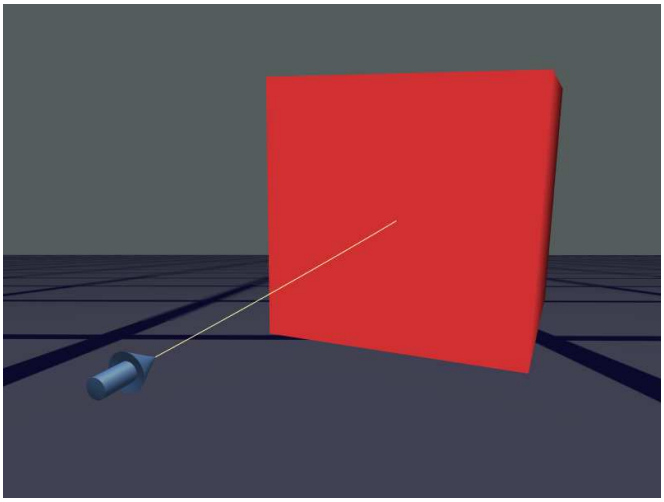


FIG. 3.18 – Une ligne tirée entre le bout du pointeur 3D et le centre du cube pour montrer la sélection de ce dernier.

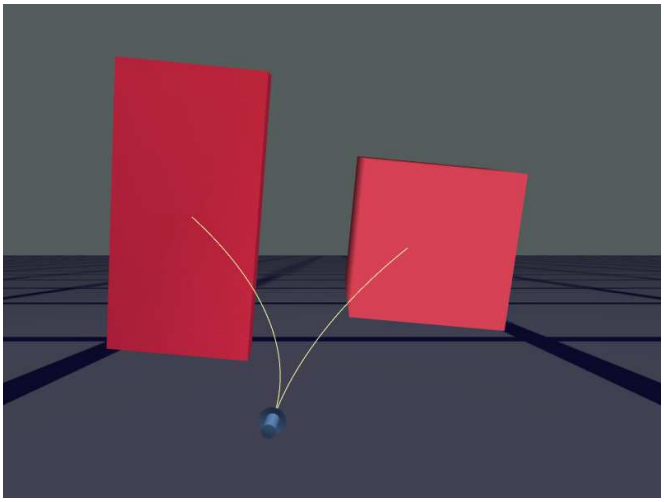
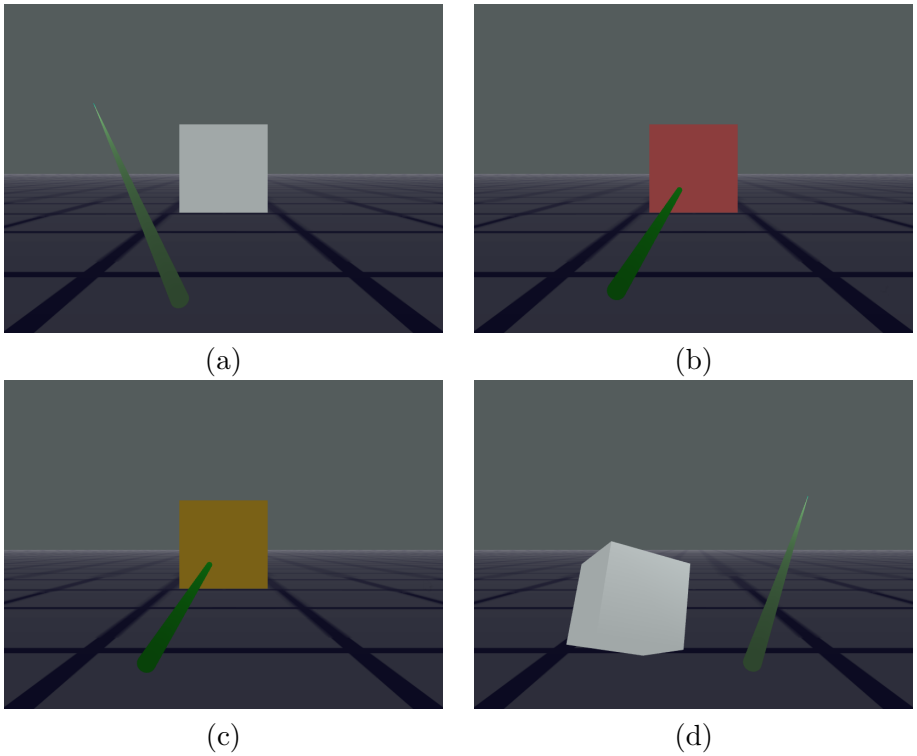


FIG. 3.19 – Un pointeur 3D manipulant deux parallélépipèdes.



(a) (b)

(c) (d)

FIG. 3.20 – Les captures d'écran montrent un rayon déplaçant un cube. (a) Le rayon virtuel est au repos. (b) Le rayon vient désigner / sélectionner le cube. L'étape de désignation est confondue avec celle de sélection dans cet exemple. (c) Le rayon manipule le cube. (d) Le rayon ne manipule plus le cube, ni ne le désigne.

Plusieurs interacteurs peuvent également manipuler simultanément un même objet grâce à des rayons virtuels. Cette situation est illustrée par la figure 3.21 qui montre aussi les limites d'un code couleur. Pour pouvoir informer un utilisateur de toutes les étapes possibles grâce à un code couleur, il faut lui demander de mémoriser un nombre important d'associations entre une couleur et un état de manipulation.

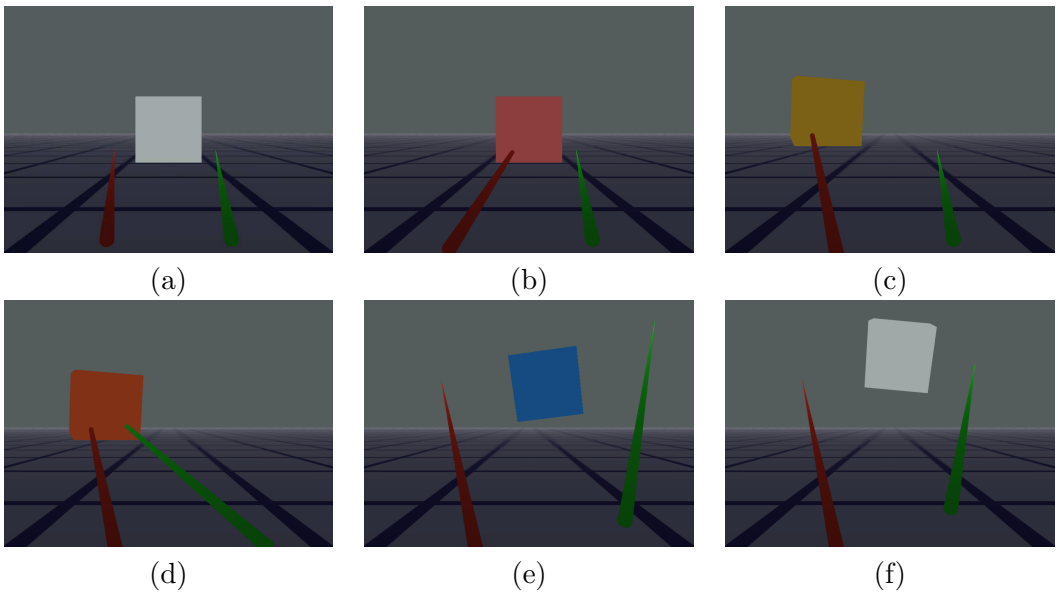


FIG. 3.21 – Les captures d'écran montrent deux rayons déplaçant un cube. (a) Les deux rayons sont au repos. (b) Le rayon de gauche désigne / sélectionne le cube. (c) Le rayon de gauche manipule le cube. (d) Le rayon de droite désigne / sélectionne le cube. (e) Le rayon de droite co-manipule le cube avec le rayon de gauche. La position du cube est la moyenne des positions proposées par les deux rayons. (f) Les deux rayons ont terminé leur manipulation.

Dans le cas où les deux rayons essaient de déplacer simultanément le cube, il devient également difficile de savoir par quoi le cube est manipulé. Pour pallier ce problème, les rayons peuvent être coudés comme montré par les figures 3.4 et 3.22. Pour des raisons propres à notre implémentation, la courbure d'un rayon est réalisée par une ligne fine. En effet, le moteur graphique employé par OpenMASK actuellement, qui est Ogre3D [Ogr], réalise une abstraction importante de la couche graphique du système d'exploitation[8] qui rend inaccessible le simple changement de l'épaisseur d'une ligne. Pour pouvoir afficher un rayon ayant un volume, la solution serait de réaliser une extrusion composée de plusieurs cylindres mis bout à bout. Pour une question de temps, cette implémentation n'a pas été réalisée.

[8]Sur Windows, DirectX et OpenGL sont utilisables. Sur Linux et MacOS X, seul OpenGL est disponible.

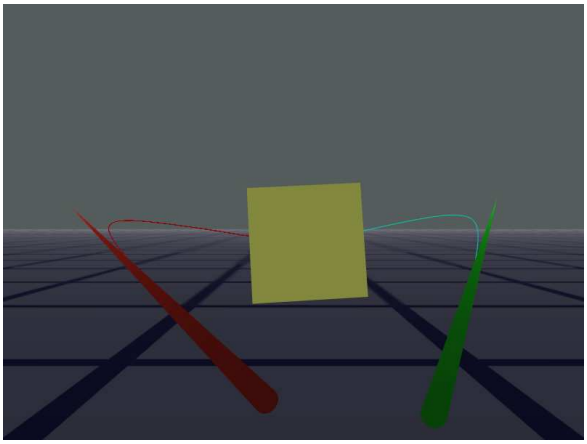


FIG. 3.22 – Deux rayons coudés co-manipulent un cube.

3.8.3 Main virtuelle

L'implémentation d'une main virtuelle peut énormément varier. Il peut s'agir de la modélisation très détaillée d'une main humaine où chaque phalange sera articulée. Dans ce cas, les difficultés sont nombreuses : maintenir des contraintes sur chaque articulation pour empêcher des mouvements impossibles par une main humaine, animer de façon réaliste les phalanges virtuelles — ce qui suppose un système de capture précis de la main — et enfin gérer le début de la manipulation lorsque les doigts viennent à la rencontre de l'objet virtuel à manipuler. Pour éviter toutes ces difficultés, une main virtuelle est souvent une sorte de pointeur : la position ouverte de la main désigne une absence de sélection/manipulation, la main fermée désigne la sélection/manipulation, enfin la main ne peut être que soit ouverte, soit fermée. La main virtuelle est alors déplaçable dans l'espace en 3D car elle est « détachée de tout bras », c'est-à-dire qu'elle n'a pas à satisfaire des contraintes réelles du corps humain. Enfin, la qualité de sa modélisation peut être faible et se résumer à un pouce accompagné d'une sorte de bloc regroupant tous les autres doigts pour simplement indiquer l'ouverture ou la fermeture de la main aux utilisateurs de l'environnement virtuel.

En pratique, nous avons choisi de ne pas implémenter une main virtuelle mais d'utiliser un pointeur virtuel. Nous lui avons toutefois ajouté la contrainte suivante : le pointeur doit venir « toucher » un objet virtuel, c'est-à-dire que sa pointe doit légèrement le traverser, pour le sélectionner.

3.9 Conclusion

Nous avons proposé un protocole d'interaction générique entre outils d'interaction et objets interactifs, indépendant de tout langage de programmation. Il permet à un outil d'interaction d'interroger un objet interactif pour connaître ses propriétés. Ensuite, un outil peut essayer d'aller modifier des propriétés. L'objet interactif, de son côté, peut

recevoir des commandes provenant, simultanément, de plusieurs outils d'interaction. C'est à l'objet interactif de gérer ces propositions multiples simultanées.

Nous sommes parvenus à implémenter ce protocole en bâtissant une architecture à base de composants logiciels que nous appelons *extensions*. Ces composants logiciels sont facilement réutilisables. Il serait possible de construire une interface graphique permettant à des non-spécialistes de venir piocher des extensions pour construire un environnement virtuel.

Par ailleurs, nous avons fourni un premier effort pour décrire les propriétés interactives d'objets interactifs au moyen du langage COLLADA.

Les implémentations ont été réalisées sur OpenMASK et Virtools[9], ce qui montre la généricité du protocole et de la description des propriétés interactives d'objets interactifs par COLLADA. Nous travaillons actuellement à une communication entre plates-formes de réalité virtuelle : OpenMASK et Virtools. L'interopérabilité pose des difficultés de communication bas-niveau : nous devons, par exemple, avoir des types de données connus de toutes les plates-formes impliquées. Mais, malgré les nombreuses difficultés d'ordre technique qui restent, nous sommes convaincus d'avoir posé un premier jalon important vers l'interopérabilité.

[9]L'implémentation sur Virtools a été réalisée par Clarté qui était partenaire dans le projet ANR Part@ge.

Chapitre 4

Interaction collaborative à trois mains

4.1 Introduction

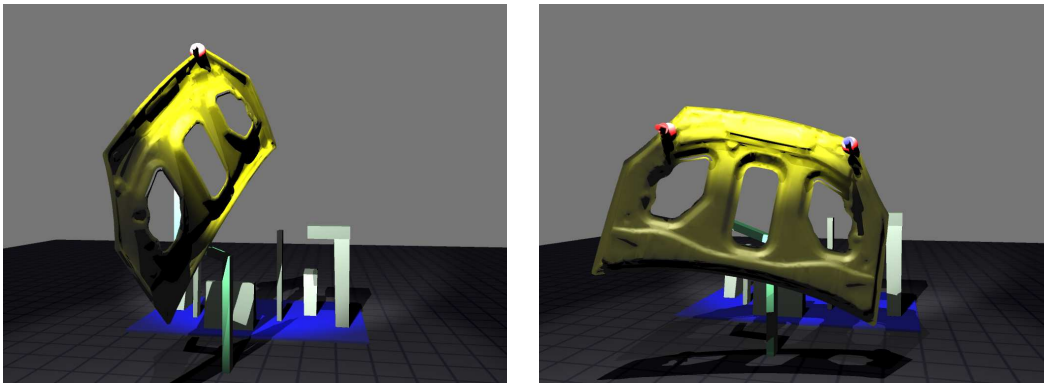
La manipulation simultanée d'un objet virtuel par plusieurs personnes dans un environnement virtuel 3D reste quelque chose de rare. Bien souvent, l'utilisateur est seul et peut manipuler un objet virtuel d'une seule main sans aucun effort : que cet objet incarne un objet réel lourd, comme une armoire, ou un objet léger, tel qu'un ballon, les mouvements sont faciles.

Nous avons souhaité intervenir sur les gestes que les utilisateurs effectuent lors de la manipulation d'un objet physique virtuel encombrant afin de les rendre plus réalistes. Nous partons d'une observation de la réalité : il est très difficile de tenir un objet encombrant avec une seule main : le poignet va d'abord fortement s'incliner puis l'utilisateur va lâcher. À deux mains, l'objet « tourne » autour de l'axe induit par les deux mains en commençant à pencher du côté le plus lourd. Pour éviter la rotation, l'utilisateur doit agripper de toutes ses forces l'objet en écartant ses doigts afin de créer un plan avec ses mains. Avec trois mains, un plan est immédiatement formé et l'objet n'oscille plus. À quatre mains, la prise est la plus naturelle parce que chaque utilisateur utilise ses deux mains.

Nous souhaitons obtenir dans l'environnement virtuel un objet qui suit un comportement similaire à celui d'un objet réel lorsqu'une seule main est employée (cf. figure 4.1 (a)), puis deux (cf. figure 4.1 (b)) et enfin trois. Le cas de la quatrième main est particulier et sera explicité.

4.2 Objectifs et limitations

Comme nous l'avons annoncé dans l'introduction, nous souhaitons imiter les gestes à appliquer à des objets encombrants virtuels afin d'inciter les utilisateurs à agir par le biais de manipulations collaboratives, via des périphériques non haptiques.



(a) (b)

FIG. 4.1 – (a) Illustration d'une manipulation à une seule « main » d'un objet virtuel (ici un capot de voiture). L'objet tourne alors autour du pointeur, qui incarne la main, comme autour d'une rotule. (b) Illustration d'une manipulation à deux mains avec deux pointeurs disposés sur deux extrémités du capot. Le capot oscille alors autour de l'axe reliant ces mains.

Par facilité, les premiers tests de manipulation ont été effectués grâce à des périphériques de bureau : un clavier, une souris 3D et des Wiimotes (manettes de Nintendo pour sa console vidéo Wii). Avec peu d'efforts, ces périphériques ont été ensuite remplacés par un mécanisme de suivi optique grâce à l'architecture sous-jacente exposée au chapitre 3. Le suivi optique a permis aux utilisateurs d'effecter des gestes plus naturels.

Par ailleurs, nous avons pensé que pour obtenir des objets virtuels donnant lieu à des mouvements proches de la réalité, il fallait les munir de propriétés physiques. Dans ce but, un moteur physique a été employé.

4.2.1 Interaction par suivi optique

Le choix du périphérique d'entrée s'est porté sur l'ARTracking qui est un système de suivi optique de la société A.R.T. [ART]. Un tel système permet un suivi aisé en position et orientation dans l'espace 3D sans que l'utilisateur n'ait à porter de fils électriques pour le relier au système de traitement (cas de capteurs magnétiques éventuellement), ni même, ici, une combinaison. Cinq caméras infrarouge ont été employées : deux étaient disposées au sol et trois au plafond. Elles capturaient la position et l'orientation à 48 Hz de cibles (cf. figure 4.2). La précision et la vitesse de reconnaissance des positions et orientations étaient suffisantes pour que l'ARTracking ne constitue pas une gêne dans l'utilisation de notre technique à 3 mains.

Par ailleurs, nous pensons que les systèmes de suivi optique sont appelés à très largement se démocratiser. En effet, Microsoft annonce, par la sortie en 2010 du projet Natal[1], un système constitué d'une caméra vidéo classique et d'une caméra infrarouge

[1]Devenu Kinect pour le nom commercial.

(a) (b)

FIG. 4.2 – (a) Exemple d'une cible dont les positions des balles réfléchissantes sont capturées par des caméras infrarouges. Cette cible était tenue à bout de doigts par un utilisateur de la technique à 3 mains. Elle pèse environ 30 g. Les autres cibles étaient quasiment identiques (dimensions, allure et poids) à celle-ci. (b) Exemple d'une caméra infrarouge utilisée.

permettant aux joueurs de sa console vidéo d'agir sans aucune manette, simplement par les mouvements de leur corps (figure 4.3). La partie logicielle ferait correspondre des squelettes pré-enregistrés aux corps des joueurs.

FIG. 4.3 – Capture de mouvements à destination du grand public avec le projet Natal de Microsoft.

De son côté, Sony proposerait, la même année, une manette (Figure 4.4) aux formes proches de celles de la Wiimote de Nintendo et comportant une sphère lumineuse sur le haut. La taille et la position de la sphère capturées par une caméra classique permettraient de déduire la position de la manette dans l'espace 3D. Enfin, des gyroscopes placés à l'intérieur de la manette fourniraient son orientation.

Cette tendance vers des interfaces de plus en plus intuitives et naturelles pour les joueurs a été initiée par Nintendo. La manette Wiimote dispose en effet d'accéléromètres

FIG. 4.4 – Capture de mouvements pour le grand public avec la manette de Sony.

et d'une caméra infrarouge, disposée à une extrémité de la manette, filmant une barre émettrice pour obtenir une position en 2D ou en 3D. La profondeur est obtenue à partir de l'écart mesuré entre les diodes infrarouges situées aux deux extrémités de la barre émettrice qui est placée sur l'écran de télévision. Si la manette est trop de biais par rapport à la barre émettrice, le calcul devient faussé. Dernièrement, cette manette s'est vue adjoindre des gyroscopes pour mieux détecter les mouvements des joueurs. Cependant, son équipement ne permet pas d'obtenir six degrés de liberté contrairement aux projets de Microsoft et Sony.

4.2.2 Intégration de calculs pour la physique

Nous souhaitons voir un objet tomber dès qu'il n'est plus manipulé. S'il est mal saisi, c'est-à-dire par une ou deux mains, il doit effectuer des mouvements de balancier comme décrit dans l'introduction de ce chapitre. Pour ce faire, nous avons employé le moteur physique Bullet [Bul]. Cette intégration est décrite dans le chapitre 6. Nous avons saisi l'occasion de l'utiliser pour détecter les collisions et empêcher les pénétrations entre objets.

4.2.3 Limites à l'interaction optique et à la physique en l'absence de retour d'efforts

Des actions sans contraintes pour l'utilisateur sur des objets physiques virtuels lui permettent d'effectuer des actions physiquement irréalistes.

D'une part, l'utilisateur peut effectuer des gestes exagérés. Par exemple, un geste ample de l'utilisateur lui permettra de lancer un objet virtuel supposé être très lourd (*i.e.* l'objet virtuel aurait une masse très grande, en tonnes par exemple) très loin sans aucune difficulté. Dans la réalité, ces gestes ne pourraient pas être effectués parce que l'objet est lourd. Avec la technique que nous proposons, l'utilisateur est actuellement obligé de « jouer le jeu » en se bornant à ne pas effectuer des gestes irréalistes (pas de grands mouvements de bras notamment) puisque nous n'utilisons pas de mécanismes à retours d'efforts pour limiter ses mouvements. Cependant, des travaux portant sur le retour haptique passif [Lec09] pourraient permettre de mieux donner à l'utilisateur une

sensation de masse pour des objets manipulés et, par conséquent, de limiter plus naturellement ses mouvements. Ainsi, il serait possible de diminuer la quantité de mouvement d'un objet manipulé par rapport aux mouvements de l'utilisateur [DLB+05] pour lui donner la sensation de manipuler un objet lourd. L'utilisateur aurait alors à effectuer des mouvements exagérés pour n'obtenir que de faibles mouvements à l'écran.

D'autre part, un utilisateur peut décider de s'écarter beaucoup trop de son partenaire durant la manipulation alors qu'un objet physique réel devrait l'en empêcher. Il est possible d'avertir l'utilisateur par des signaux visuels comme cela sera exposé. Si toutefois l'utilisateur réussissait à s'écarter de son partenaire dans la réalité, alors l'objet réel manipulé serait en fait déformé ou bien endommagé.

4.3 Techniques de manipulation collaboratives existantes

La technique présentée dans ce chapitre fait intervenir trois mains provenant de deux ou trois utilisateurs. Par conséquent, cette technique s'appuie sur des résultats issus de travaux concernant des interactions collaboratives et également des interactions faisant intervenir deux mains. Ces domaines ont été abordés au cours de l'état de l'art au chapitre 1 et ne seront donc repris que brièvement.

4.3.1 Techniques à deux mains

Cutler et *al.* [CFH97] décrivent une manipulation à deux mains sur le « Responsive Workbench » (cf. chapitre 1). L'une des commandes offertes par cette interface est la technique « grab-and-carry »[2] qui fournit cinq degrés de liberté. Par le biais de cette technique bimanuelle symétrique, l'utilisateur peut tourner et déplacer un objet virtuel avec ses deux mains. La technique nommée « grab-and-twirl »[3] étend la technique « grab-and-carry » en donnant accès à l'utilisateur à un sixième degré de liberté qui est celui induit par le segment reliant les deux mains de l'utilisateur. Enfin, et contrairement aux deux précédentes, la technique « trackball » est bimanuelle asymétrique pour permettre aux utilisateurs d'utiliser leur main non-dominante pour positionner un objet pendant que leur autre main oriente cet objet en son centre.

Selon nous, aucune de ces techniques n'apparaît réaliste. En effet, le positionnement et l'orientation d'un objet devrait donner à l'utilisateur l'impression de tenir l'objet dans le creux de ses mains. Pour cela, d'une part l'utilisateur devrait toujours être contraint de garder ses mains proches de l'objet virtuel[4], d'autre part ses mains devraient effectuer des actions symétriques.

4.3.2 Techniques collaboratives

Plusieurs méthodes sont disponibles pour combiner des actions provenant de plusieurs utilisateurs afin d'obtenir les mouvements de l'objet virtuel manipulé [RSJ02].

[2] « Saisir et transporter » en français.
[3] « Saisir et pivoter » en français.
[4] La co-localisation est ici essentielle.

Une première approche consiste à effectuer une moyenne des deux mouvements. La moyenne peut être utilisée pour la métaphore du Bent Pick Ray [RHWF06]. De même avec la technique du SkeweR [DLT06], mais la détermination d'une rotation autour de deux « points de fixation » est problématique et doit être gérée de façon non-naturelle. Si deux mains virtuelles sont employées pour la construction d'un belvédère virtuel [RWOS03a], la manipulation d'une poutre virtuelle par le biais de l'interpolation des deux rotations risque de la vriller.

Une deuxième approche consiste à ajouter les deux mouvements (intégration asymétrique des mouvements). Cette approche peut être associée à une distribution préalable des degrés de liberté aux utilisateurs [PBF02]. Par exemple, un utilisateur manipule uniquement les translations tandis que l'autre ne manipule que les rotations.

Enfin, une troisième approche consiste à ne garder que la partie commune (intersection) de deux mouvements (intégration symétrique de mouvements). Mais aucune des approches ne semble idéale. En fait, la technique d'intersection est la plus adaptée lorsque deux utilisateurs ont à effectuer une action très similaire, tandis que la technique de la moyenne est préférée par les utilisateurs lorsqu'ils doivent effectuer des mouvements légèrement différents.

4.3.3 Conclusion

Les techniques exposées permettent effectivement une manipulation à deux personnes, ou à une personne avec deux mains, mais probablement aucune ne donne l'impression réaliste de manipuler un objet réel au travers de la manipulation d'un objet virtuel. Cette situation est le point de départ de la technique à trois mains que nous proposons.

4.4 La technique de manipulation à trois mains

Nous allons à présent expliquer le principe de la technique de manipulation à trois mains que nous proposons [ADL09]. Nous donnerons également des résultats d'observations informelles que nous avons effectuées.

4.4.1 Principe

La technique de manipulation à trois mains permet de manipuler dans un espace en 3D des objets virtuels. Les six degrés de liberté de l'objet manipulé sont obtenus au moyen des seules positions de trois points non-alignés. Ces points sont contrôlés par les mains des utilisateurs.

La figure 4.5 montre deux utilisateurs en train de manipuler un capot virtuel de voiture[5]. L'utilisateur de gauche controle deux points de manipulation à la fois tandis que l'autre utilisateur est limité à un seul. Les positions de leurs mains réelles sont suivies

[5]Plus précisément, il s'agit du dessous du capot d'une voiture avant soudure, d'où ses trois grands trous.

par un système de suivi optique infrarouge qui repère les marqueurs que les utilisateurs tiennent au bout de leurs doigts. Les résultats de leur manipulation apparaissent à l'écran disposé en face d'eux. La figure 4.6 montre quant à elle trois utilisateurs en train de manipuler le capot affiché à l'écran. Cette fois, chaque utilisateur ne peut utiliser qu'une seule main.

4.4.2 Manipulation et retour visuel à l'utilisateur

Dans les travaux que nous avons présentés, les mains des utilisateurs ont été représentées à l'écran par des pointeurs. Un lancer de rayon est effectué en permanence pour détecter lorsqu'une main est suffisamment proche de l'objet à manipuler. Ce rayon émane de la pointe du pointeur et suit sa direction mais pourrait, par exemple, provenir de la paume d'une main virtuelle si la géométrie d'une main était utilisée. De plus, le rayon fournit un point d'intersection avec l'objet à manipuler qui est appelé *point de manipulation* (cf. figure 4.7). Si un utilisateur démarre une manipulation, une sphère virtuelle est affichée pour indiquer la position du point de manipulation.

Par ailleurs, un ruban élastique est ajouté entre un point de manipulation et la main qui le manipule (cf. figure 4.8) pour deux raisons. D'abord, le ruban indique clairement la main propriétaire (*i.e.* manipulatrice) d'un point de manipulation pour aider les utilisateurs à comprendre qui agit sur quoi. Ensuite, le ruban essaie de mieux faire prendre conscience aux utilisateurs de la distance entre leurs mains virtuelles et les points de manipulation. Dans la réalité, un tel écart aux points de manipulation indiquerait une déformation de l'objet, ou bien sa rupture, ou bien un glissement des points d'accroche. Dans le monde virtuel proposé, l'impossibilité de déformer l'objet virtuel manipulé, parce que nous travaillons sur des objets rigides, conduit à une instabilité des calculs effectués par le moteur physique qu'il faut éviter. Le ruban arbore un code couleur où un rouge de plus en plus vif signale à un utilisateur qu'il doit rapprocher sa main virtuelle du point de manipulation correspondant. C'est par ce moyen que les utilisateurs sont, finalement, invités à rester assez proches de leur partenaire au cours d'une manipulation puisque chacun reste proche de l'objet virtuel manipulé.

4.4.3 Calcul du déplacement de l'objet manipulé

Les mouvements de l'objet manipulé peuvent être calculés de différentes façons en utilisant les positions fournies par les trois mains des utilisateurs. Une solution consiste à conserver les points de manipulation les plus proches possibles des mains. Au début de la manipulation, les positions des mains notées H_1, H_2 et $H3$, correspondent aux positions de leurs points de contacts P_1, P_2 et P_3, avec l'objet manipulé. Ces points de contact sont les points de manipulation (cf. figure 4.9).

Lorsque les utilisateurs déplacent leurs mains en H'_1, H'_2, H'_3, la translation $\vec{T}$ à appliquer à la position initiale P_c de l'objet manipulé est calculée de la façon suivante :

$$H_0 = \frac{H_1 + H_2 + H_3}{3};\ H'_0 = \frac{H'_1 + H'_2 + H'_3}{3};\ \vec{T} = \overrightarrow{H_0 H'_0}.$$

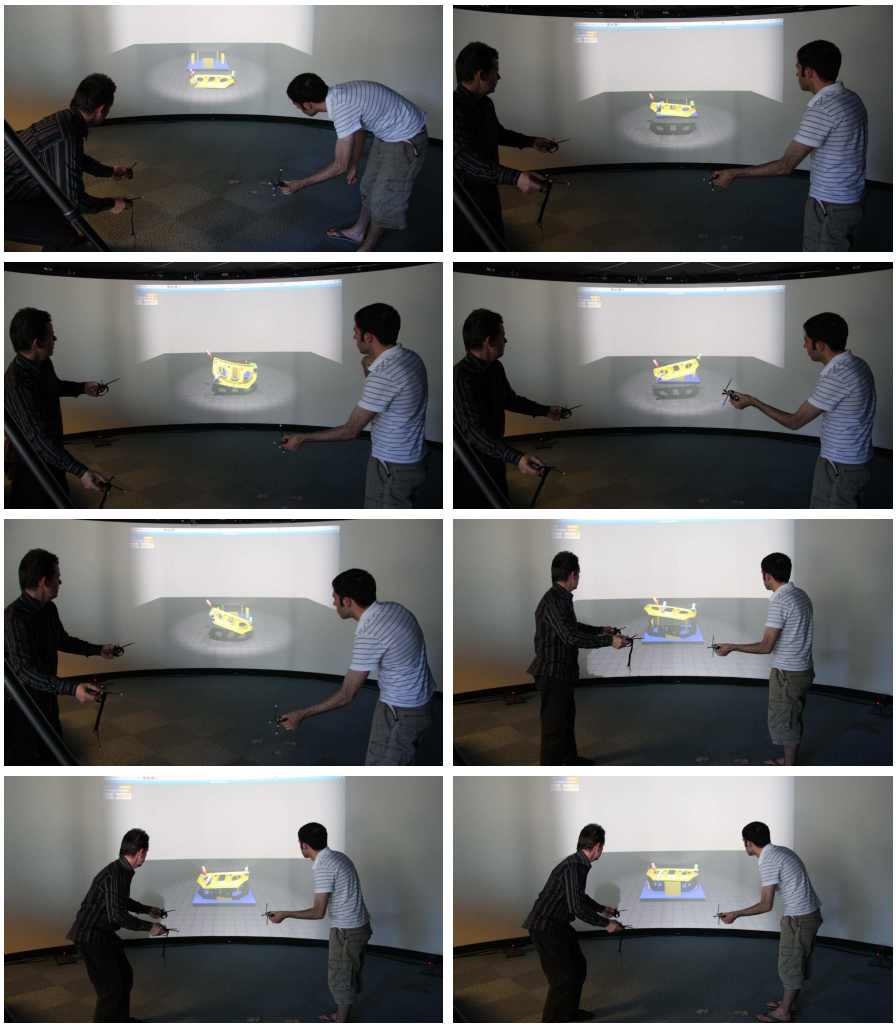


FIG. 4.5 – Séquence de manipulation à deux utilisateurs par la technique à trois mains. On observe notamment que l'utilisateur qui utilise ses deux mains peut faire varier l'assiette du capot autour de l'axe qui le relie à l'autre utilisateur. Par ailleurs, chaque utilisateur peut faire varier l'assiette autour de l'axe qui sépare les deux utilisateurs : l'utilisateur à une main monte ou descend sa main tandis que l'utilisateur à deux mains doit les monter ou les descendre simultanément.

FIG. 4.6 – Exemple d'une manipulation à trois utilisateurs où chacun n'utilise qu'une seule main.

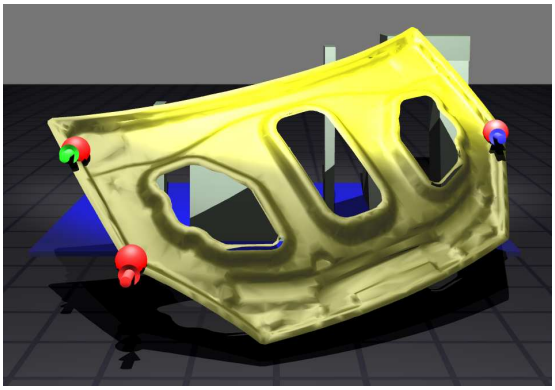


FIG. 4.7 – Illustration d'une manipulation à trois mains : trois pointeurs (vert, rouge et bleu) les représentent. Ils sont tous trois proches de leurs points de manipulation respectifs qui sont représentés par des sphères rouges.

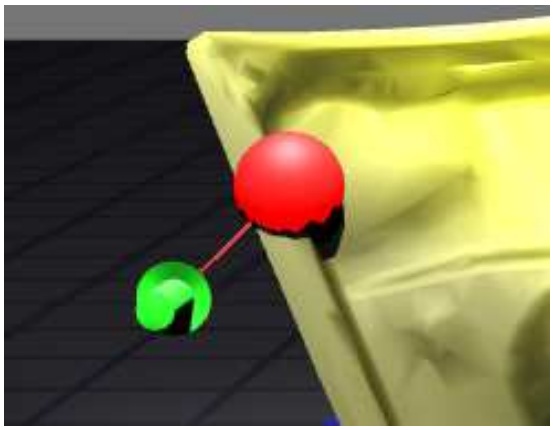


FIG. 4.8 – Illustration du ruban élastique apparaissant entre un point de manipulation et un pointeur. L'écart entre un pointeur et son point de manipulation a, volontairement, été exagéré ici pour illustration.

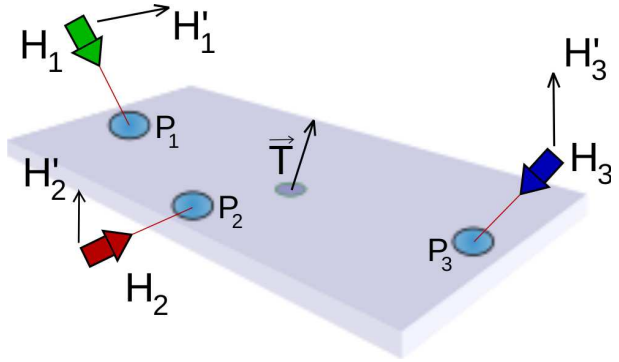


FIG. 4.9 – Calcul du déplacement $\vec{T}$ d'un objet virtuel au moyen des mouvements de trois mains.

Afin de calculer la différence de rotation avec l'orientation initiale du plan, la première étape implique le calcul de la rotation $(\vec{R_1},\ \alpha_1)$ qui transforme $\vec{i}$ en $\vec{i'}$ (cf. figure 4.10) :

$$\vec{i} = \overrightarrow{H_0H_2};\ \vec{j} = \overrightarrow{H_0H_3} \quad ;\quad \vec{i'} = \overrightarrow{H'_0H'_2};\ \vec{j'} = \overrightarrow{H'_0H'_3}$$
$$\vec{R_1} = \vec{i} \times \vec{i'} \quad ;\quad \alpha_1 = \arccos\left(\frac{\vec{i}\cdot\vec{i'}}{\|\vec{i}\| * \|\vec{i'}\|}\right).$$

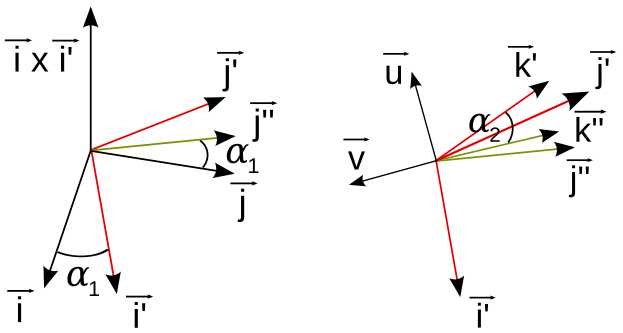


FIG. 4.10 – Rotation $(\vec{R_1}, \alpha_1)$ et rotation $(\vec{i'}, \alpha_2)$.

Cette rotation transforme $\vec{j}$ en un vecteur intermédiaire $\vec{j''}$:

$$\vec{j''} = (\vec{R_1}, \alpha_1) * \vec{j}.$$

La seconde étape consiste en le calcul de la rotation $(\vec{i'},\ \alpha_2)$ qui transforme $\vec{j''}$ en $\vec{j'}$. Après la construction d'une nouvelle base orthogonale 3D $(\vec{u},\ \vec{v},\ \vec{i'})$, cet angle est calculé

comme suit :

$$\begin{aligned} \vec{k''} &= (\vec{j''} \cdot \vec{u}) * \vec{u} + (\vec{j''} \cdot \vec{v}) * \vec{v} \\ \vec{k'} &= (\vec{j'} \cdot \vec{u}) * \vec{u} + (\vec{j'} \cdot \vec{v}) * \vec{v} \\ \alpha_2 &= \arccos\left(\frac{\vec{k''} \cdot \vec{k'}}{\|\vec{k''}\| * \|\vec{k'}\|}\right). \end{aligned}$$

La différence entre l'orientation actuelle et celle initiale de l'objet est obtenue en combinant ces deux rotations : $(\vec{i'}, \alpha_2) * (\vec{R_1}, \alpha_1)$.

Si le premier triangle défini par les positions des trois mains et le second défini par les positions initiales des trois points de manipulation ne conservent pas la même forme, alors l'angle de la rotation autour de l'axe orthogonal à ces deux triangles est la meilleure approximation possible. Dans le cas contraire, cet angle est exact.

Une autre solution d'implémentation est celle décrite dans le chapitre 6. Des contraintes ponctuelles d'un moteur physique, tel que Bullet [Bul] ou NVIDIA PhysX [Phy], sont employées. Une contrainte est alors ajoutée dynamiquement entre une main virtuelle et un point de manipulation.

Dans les deux cas d'implémentation, l'utilisation de rubans colorés peut aider les utilisateurs à maintenir leurs mains proches des points de manipulation.

4.5 Premières observations informelles

La technique des trois mains de manipulation n'a pas été évaluée de façon rigoureuse. Après l'avoir fait tester à cinq personnes, les premiers avis montraient qu'effectivement cette technique procure un sentiment de réalisme à ses utilisateurs mais tout en nécessitant un temps d'apprentissage important. Ces premiers testeurs appréciaient le cas où ils pouvaient utiliser leurs deux mains pour manipuler l'assiette de l'objet virtuel (qui était un capot de voiture comme sur les illustrations). Le cas où ils étaient limités à l'usage d'une seule main les frustrait un peu parce qu'ils gardaient une main ballante, non utilisable.

En tant qu'auteurs, nous pensons que cette technique requiert une période d'apprentissage importante. Nous pensons cela parce que c'est ce que nous avons nous-même ressenti : nos premiers tests étaient un peu maladroits. En particulier, nous n'osions pas faire faire de grandes translations ni des rotations très prononcées. Nos premiers testeurs suivaient d'ailleurs un comportement similaire. C'est après plusieurs essais que l'utilisateur réussit à intégrer différentes observations lui permettant de mieux agir.

Nous pensons cependant que notre technique n'est pas, pour autant, difficile à utiliser. En effet, au cours de diverses démonstrations et de salons, un public très varié composé de spécialistes comme de non-spécialistes de la réalité virtuelle a montré qu'il était facile de s'approprier cette technique en quelques minutes.

C'est également en faisant tester notre technique à un public large que nous avons pu effectuer les observations suivantes. Il y eu 3 jours de démonstrations pendant la conférence JVRC 2009 [ADLA09] à Lyon, puis des essais en alternance avec l'interface

tangible triangulaire proposée au chapitre 5 pendant 3 jours au colloque STIC 2010 à Paris, ou encore à la journée portes-ouvertes de l'INSA de Rennes en 2010.

L'usage des contraintes est robuste. Le système de trois contraintes point à point de Bullet s'avère robuste : il occasionne très peu de mouvements imprévisibles de l'objet qui seraient dûs à des instabilités, même si les utilisateurs écartent très franchement leurs mains virtuelles des points de manipulation. D'autre part, des translations rapides ou de grandes rotations (de 360 degrés, par exemple) sont tout à fait applicables. Par conséquent, notre système bride peu les mouvements des utilisateurs.

La coordination des deux mains est à apprendre. Dans le cas d'une configuration d'un utilisateur employant ses deux mains avec un autre utilisateur à une seule main, le temps d'adaptation de celui qui utilise ses deux mains peut être important. En effet, il lui faut apprendre à synchroniser ses mouvements. Ainsi, pour incliner un objet, il doit penser à monter l'une de ses mains et, dans le même temps, à faire descendre l'autre. Cet apprentissage n'est pas immédiat même s'il nous a semblé rapide à surmonter.

Il faut dialoguer avec l'autre utilisateur. Les manipulations ont rapidement fait apparaître un comportement assez individualiste chez plusieurs testeurs. Leurs interactions apparaissent comme une séquence en deux parties.

La première partie consiste à effectuer les mouvements adéquats pour déplacer l'objet manipulé. Durant cette phase, l'utilisateur semble ne pas regarder son partenaire et agit en étant convaincu que ses mouvements sont les bons. Dans les faits, il est peut-être en train d'essayer de faire passer l'objet à travers un autre objet, ou l'autre utilisateur est peut-être en train d'effectuer un mouvement complètement opposé ! Il semblerait que, durant cette phase, un utilisateur de ce type ne soit intéressé que par les positions de ses pointeurs.

La deuxième partie de la séquence est le moment où l'utilisateur semble enfin s'interroger sur les actions de son partenaire. Parfois, l'utilisateur peut alors devenir agacé parce que son partenaire ne l'a pas suivi. C'est à ce moment que, en attendant son partenaire, le premier utilisateur ne regarde plus du tout ses actions et peut se retrouver à faire des mouvements involontaires (baisser ses bras, par exemple) en voulant se mettre à la place de l'autre et en oubliant son propre rôle.

Ces agissements montrent qu'il faut apprendre à travailler en collaboration. Chez ces personnes, la quantité de communication était faible. Durant la première phase que nous avons décrite, il n'y avait quasiment pas d'échange. La deuxième phase était marquée par un agacement plus ou moins important. Seuls les groupes très communicants étaient rapidement efficaces.

4.6 Conclusion et perspectives

D'abord, notons que cette technique sera très probablement une technique facile à mettre en œuvre grâce à une large diffusion de systèmes de suivi optique à bas-coût pour le grand public. Ensuite, elle nous paraît intéressante parce que, par rapport aux techniques collaboratives de manipulation actuelles, elle autorise les utilisateurs à faire des gestes plus réalistes. Cependant nous avons pu remarquer que les deux principales difficultés qu'elle impose sont : la contrainte de non-éloignement des mains pour chaque utilisateur d'une part, et la contrainte de suivi des actions effectuées par le partenaire d'autre part. Nous avons souhaité apporter une réponse au dernier problème grâce à l'usage d'une interface tangible qui va fixer les positions des trois mains les unes par rapport aux autres. Cette solution est présentée au chapitre 5.

Comme nous l'avons évoqué, nous pourrions découpler les mouvements des mains réelles des utilisateurs par rapport à leurs mains virtuelles pour faire percevoir aux utilisateurs le poids des objets réels correspondant aux objets virtuels. En pratique, les utilisateurs auraient à faire plus de mouvements, par exemple à lever plus haut leur bras que de raison, pour déplacer l'objet virtuel « lourd » manipulé.

Par ailleurs, nous souhaiterions apporter des aides aux utilisateurs par des mécanismes de guidage qui reposeraient sur la limitation des mouvements possibles. Par exemple, nous pensons qu'il pourrait être intéressant dans des situations de manipulation pour ajustements que l'utilisateur à deux mains ne puisse qu'appliquer des mouvements verticaux à l'objet manipulé. Dans ce cas, la rotation autour de l'azimut par rapport au sol deviendrait impossible. La réalisation de ce scénario passerait par le remplacement d'une contrainte de type rotule au profit d'une contrainte de type charnière au niveau du point de manipulation de l'utilisateur à une main. En suivant cette idée de remplacer les contraintes rotule par d'autres types de contraintes, nous pensons, par exemple, qu'une contrainte de type glissière pourrait aider l'utilisateur à une main. Pour cela, l'axe de glissement serait la droite reliant les points de manipulation de l'utilisateur à deux mains. Ainsi, l'utilisateur à une main pourrait faire glisser à l'horizontal l'objet manipulé pour, peut-être, l'encastrer plus facilement dans un autre objet virtuel.

Enfin, des guidages pourraient s'accompagner de mouvements automatiques des points de manipulation. Par exemple, lorsque l'utilisateur à deux points de manipulation avancerait, le point de manipulation restant le suivrait en conservant une hauteur fixée. Par ce biais, un utilisateur seul pourrait simuler la présence d'une deuxième personne et estimer la manipulation qui serait obtenue avec deux utilisateurs.

Chapitre 5

Une interface tangible reconfigurable pour la manipulation collaborative

5.1 Introduction

Le chapitre 4 a présenté une technique d'interaction permettant à deux ou trois utilisateurs de co-manipuler simultanément un objet virtuel grâce à trois mains. La configuration de l'ensemble de ces mains doit être conservée au cours de la manipulation pour correctement imiter la réalité et parce que l'objet manipulé n'était pas déformable. Pour aider les utilisateurs à conserver une configuration, nous proposons un nouveau concept d'Interface Tangible Reconfigurable [ADL10b]. En anglais, nous avons nommé cette interface tangible « Reconfigurable Tangible Device », d'où l'abréviation « RTD » utilisée dans ce chapitre.

Ce chapitre présente deux instances du RTD : une version à 3 points de manipulation (RTD-3) et une version à 4 points de manipulation (RTD-4). Le RTD-3 a été évalué en le comparant à des techniques d'interaction collaboratives : la Moyenne des positions et orientations fournies par des utilisateurs et la Séparation des degrés de libertés en fonction des utilisateurs.

5.2 L'interface tangible reconfigurable

Le but de l'interface tangible reconfigurable (RTD) est de proposer une interface physique qui corresponde à de nombreuses formes d'objets 3D pour leur manipulation. Cette interface doit être facilement et rapidement reconfigurable pour éviter des interruptions trop longues de l'activité de ses utilisateurs. Elle est rigide pour fournir un retour haptique passif entre les mains qui l'emploient.

Le RTD est un nouveau concept d'interface tangible reconfigurable qui propose une interface physique générique et universelle comprenant des points rigidement liés entre eux. Ces points de manipulation sont des *poignées* qui constituent une forme simple

pouvant être modifiée. La forme obtenue à partir des points esquisse la forme (ou le maillage) de l'objet virtuel manipulé par ses utilisateurs. Les liens physiques entre les points sont rigides mais faciles à étirer ou raccourcir. Cette interface a été pensée dans le but de permettre à deux utilisateurs (ou plus) d'interagir simultanément sur un objet virtuel.

5.2.1 Domaines d'application

Nous visons les domaines de l'apprentissage dans des environnements virtuels, du prototypage virtuel et, éventuellement, du ludique.

Plus précisément, nous souhaitons rendre les gestes des utilisateurs les plus naturels possibles au sein d'environnements virtuels, c'est-à-dire les plus proches possibles de la réalité. Nous espérons alors qu'en permettant à des personnes de faire des gestes fidèles à la réalité dans un environnement virtuel, elles les apprendront et sauront les reproduire dans un environnement réel.

Nous pensons également que des mouvements les plus réalistes possibles dans le virtuel permettront de mieux détecter des erreurs de conception industrielle dans les phases de prototypage virtuel, ou bien de mieux envisager des méthodes d'amélioration des tâches à effectuer dans le réel.

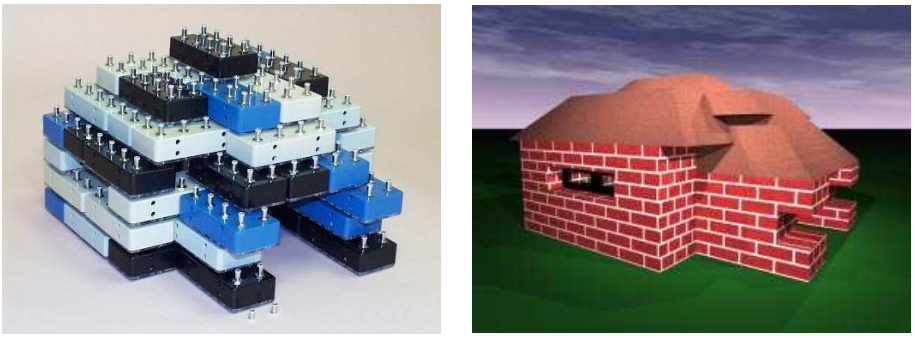
Enfin, l'aspect naturel et intuitif de l'interface tangible pourrait permettre un accès rapide au grand public. Il est alors possible d'envisager son usage pour des jeux vidéos.

5.2.2 Intérêts d'une interface tangible

Une interface tangible est conçue dans le but de donner une forme physique à une information numérique [UI00]. Une telle interface peut alors donner l'impression à ses usagers qu'ils tiennent l'objet virtuel dans leurs mains, ou du moins, une partie de l'objet virtuel. Cette idée a été introduite en réalité virtuelle par Hinckley et *al.* [HPGK94] pour une application de neurochirurgie comme nous l'avons mentionnée dans la partie 1.3 de l'état de l'art. Avec celle-ci, l'interacteur effectue des coupes pour voir les résultats d'une radiographie d'un patient (cf. figure 1.11). Récemment, Salzmann *et al.* [SJF09] ont proposé d'utiliser une interface tangible qui a la forme d'une barre. Elle relie les mains des deux utilisateurs qui l'emploient, ce qui la met en rapport avec la technique des trois mains et l'interface tangible reconfigurable que nous proposons. Dans cette tâche, chacun des deux utilisateurs n'utilise qu'une seule de ses mains pour manipuler le pare-brise virtuel d'une voiture. Ensemble, les utilisateurs amènent le pare-brise d'une position initiale à son emplacement sur la carrosserie. Les auteurs ont constaté que la manipulation sans leur interface tangible était plus longue que celle avec l'interface.

Diverses interfaces ont été développées pour imiter les formes d'objets à manipuler (pas nécessairement en réalité virtuelle). Nous qualifions ces interfaces tangibles de reconfigurables, par opposition à celles dont la forme ne peut être changée comme [HPGK94] et [SJF09]. Une première approche pour approximer la forme d'un objet à manipuler consiste à le construire par un assemblage de briques. Cette approche a donné lieu à de nombreux travaux de recherche qui ont débuté par une application architec-

turale [Fra95] pour ensuite donner lieu, par exemple, aux briques de MERL [AFM+99] (cf. figure 5.1) ou aux ActiveCubes [WIA+04, IIKK04] (cf. figure 5.2). Une deuxième approche consiste à modifier la forme d'interfaces tangibles qui sont malléables. Par exemple, l'utilisateur applique des pressions avec ses doigts sur l'interface proposée dans [STP08] pour modéliser des objets en 3D. Dans Glume [PLI06], des sphères malléables sont reliées entre elles par une structure rigide sous-jacente. Dans Senspectra [LPI07], des balles rigides sont connectées ensemble par des flexibles pour constituer une structure globale flexible. La figure 5.3 illustre ces deux interfaces tangibles.



(a) (b)

FIG. 5.1 – (a) Un assemblage des briques de MERL [AFM+99]. (b) Un objet virtuel manipulé par les briques de (a), celles-ci approximent la forme de ce bâtiment.

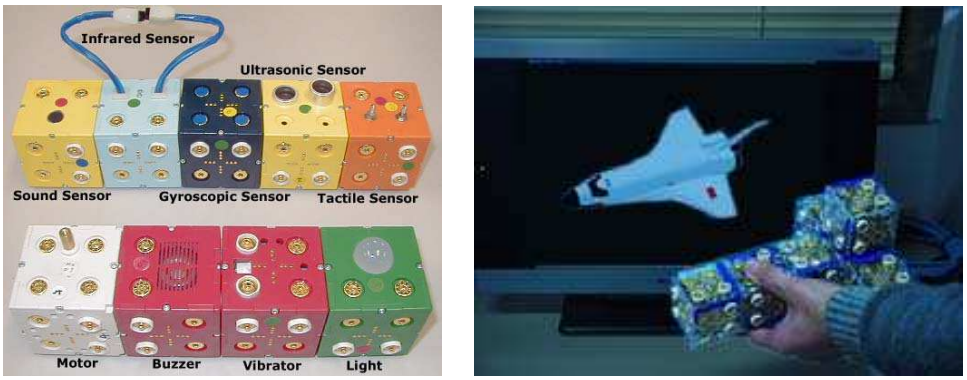
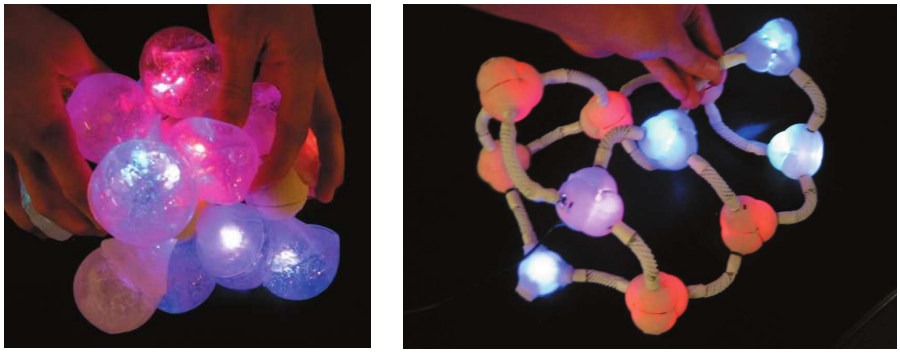


FIG. 5.2 – (a) Différents types d'ActiveCubes [WIA+04]. Cette illustration montre que ces cubes peuvent embarquer différentes fonctionnalités telles que l'émission de sons ou de lumières, ou bien la récupération d'orientations par des gyroscopes. (b) Des ActivesCubes assemblés approximent la forme d'une navette spatiale virtuelle qu'un utilisateur peut orienter à l'écran.

Toutefois, même si les utilisateurs semblent préférer les interfaces tangibles à celles non tangibles, les performances observées ne sont pas toujours meilleures [HTP+97,



(a) (b)

FIG. 5.3 – (a) Illustration de l'interface tangible Glume [PLI06]. (b) Illustration de l'interface tangible Senspectra [LPI07].

WR99]. Malgré cela, le retour tactile passif procuré par une interface tangible peut améliorer l'impression d'immersion ainsi que l'efficacité d'un entrainement au sein d'un environnement virtuel [IMWB01]. Dans le cas d'interactions collaboratives, ce retour haptique passif procuré par une interface tangible peut améliorer les performances [SJF09].

5.2.3 Un triangle reconfigurable pour des manipulations à trois points

Le triangle est une instance du RTD où 3 points de manipulation sont utilisés (cf. figure 5.4). Cette version du RTD est nommée *RTD-3*. Le nombre de points trouve son origine dans la technique d'interaction à 3 mains présentée au chapitre 4. En effet, il s'agissait d'abord d'aider les manipulateurs à conserver leurs mains à une distance constante durant une manipulation à 3 mains. Pour cette raison, l'instance du RTD qui est présentée hérite des avantages de la technique à 3 mains. Notamment, la position et l'orientation de l'objet virtuel manipulé sont obtenues à partir des positions des 3 mains uniquement.

FIG. 5.4 – Illustration de l'interface tangible reconfigurable triangulaire. L'utilisateur à gauche tient le RTD-3 par ses deux mains, aux coins, tandis que l'autre utilisateur n'en emploie qu'une seule.

L'usage de trois points présente l'avantage de définir un plan par le nombre minimal possible de points. Avec ce faible nombre, le triangle reste facile à attacher à un objet à manipuler. La séquence de manipulation est comme suit :

1. les trois points virtuels manipulés par le triangle doivent venir effleurer l'objet virtuel à manipuler. Il s'agit ici d'imiter une préhension de l'objet avec les mains ;
2. un utilisateur déclenche le début de la manipulation. Une télécommande peut être utilisée pour un déclenchement distant (nous avons, par exemple, utilisé une Wiimote de Nintendo), ou bien un opérateur peut employer le clavier.

La possibilité de reconfigurer la forme du RTD-3 aide également à accomplir plus rapidement et plus aisément l'étape 1. En effet, les trois branches du triangle sont télescopiques et peuvent être allongées ou raccourcies après avoir libéré le coulissement par un bouton.

Le RTD-3 s'adapte aux petits objets comme aux grands grâce à une variation importante des angles : d'environ 20° à 130°, grâce à des branches dont la longueur, pour chacune, s'étend de 38 cm à 95 cm. La figure 5.5 montre que ces différentes variations permettent la manipulation d'objets longs ou courts, aux formes rondes, cubiques, etc. de façon horizontale ou verticale. Pour un petit objet, comme une balle, les branches du triangle seront raccourcies pour atteindre la longueur minimale. Pour un carton plus grand que la balle, le triangle peut être légèrement agrandi. Enfin, un objet de forme allongée, telle une lampe, peut être contrôlé par un triangle isocèle dont un flanc est « écrasé ».

5.2.4 Réalisation de l'interface à trois points

Nous souhaitions, dès le départ, que l'interface tangible triangulaire ait des branches télescopiques. Un premier prototype s'est basé sur six antennes télescopiques qui sont habituellement employées sur des postes radiophoniques (cf. figure 5.6). Chacune des trois branches du triangle était formée par deux antennes en sens opposés afin de rigidifier la structure. Le triangle ainsi obtenu était très léger et très compact (chaque branche mesurait 10 cm) tout en offrant une taille assez grande une fois déployé (dans ce cas, chaque branche mesurait 70 cm). Malheureusement, la rigidité de l'ensemble n'était pas satisfaisante. Avec des utilisateurs crispés effectuant des actions divergentes, le triangle se tordait très rapidement et offrait un lien haptique trop faible entre les deux utilisateurs.

La deuxième version du RTD-3 est celle utilisée actuellement et illustrant ce chapitre. Ses branches sont issues d'un trépied pour appareil photographique qui a été démonté. L'ensemble reste léger tout en devenant rigide et robuste.

Pour les deux versions du RTD-3, les marqueurs optiques ont été placés dans ses coins. Nous avons fait ce choix pour que sa manipulation reste identique à celle de la technique à trois mains. Certains utilisateurs auraient souhaité pouvoir placer les capteurs aux centres des branches mais nous n'avons pas encore évalué l'apport éventuel. Les marqueurs sont ceux utilisés lors de la manipulation à trois mains. Afin de les fixer, trois petites surfaces (moins de 2 cm^2 chacune) en carton ont été placées au dessus de

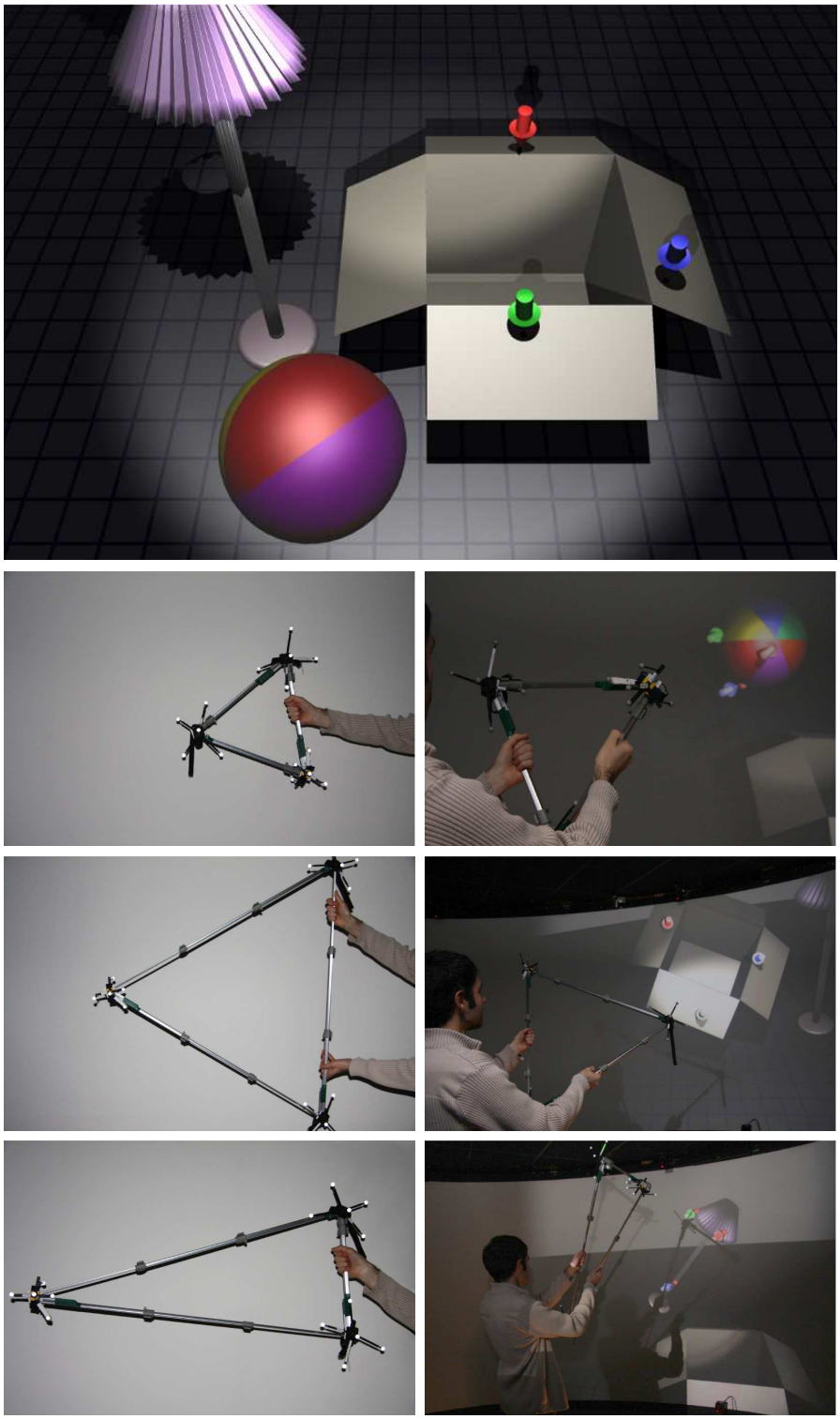


FIG. 5.5 – L'image supérieure montre l'ensemble d'une scène virtuelle de test. Les deux colonnes montrent différentes formes du RTD-3 pour correspondre à l'objet virtuel à manipuler : une balle, une boîte en carton et une lampe.

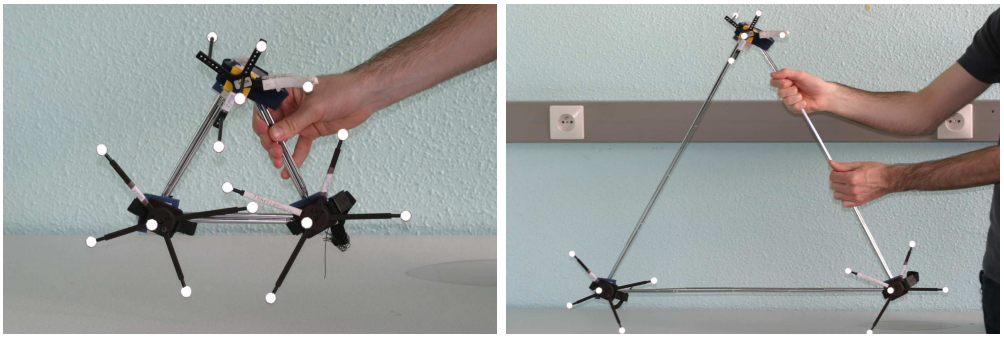


FIG. 5.6 – Premier prototype du RTD-3. Ce triangle est composé d'antennes télescopiques. Ces illustrations le montrent sous deux tailles : raccourcie au maximum et étendue au maximum.

chacun des coins en veillant à ne pas bloquer les articulations. Un marqueur est alors accroché à cette surface par une bande auto-agrippante et peut aisément être remplacé si besoin.

5.3 Évaluation de l'interface tangible reconfigurable triangulaire

Nous souhaitions comparer le RTD-3 à des techniques de manipulation collaborative couramment rencontrées en réalité virtuelle. À travers une tâche de type « pick-and-place[1] », des couples d'utilisateurs ont évalué notre technique ainsi que celle de la Moyenne et de la Séparation des degrés de liberté où toutes deux n'utilisaient pas une interface tangible [ADL10a]. Dans les trois cas, il s'agissait de déplacer un capot virtuel de voiture à deux utilisateurs. Différentes données ont été relevées telles que le temps de manipulation et le nombre de collisions du capot avec l'environnement virtuel. Les utilisateurs ont également rempli des questionnaires évaluant leurs préférences subjectives.

5.3.1 Descriptions des techniques à comparer

5.3.1.1 La moyenne

La moyenne vise à combiner les mouvements de plusieurs utilisateurs en calculant des moyennes sur les positions et les orientations que chacun suggère. Cette technique est très souvent employée pour combiner des actions en réalité virtuelle afin de déplacer un objet à plusieurs utilisateurs. Elle trouve une description, par exemple, dans [RSJ02].

Nous avons décidé de baser notre implémentation sur les mouvements des utilisateurs au lieu des positions qu'ils donnent directement. Par conséquent, les utilisateurs

[1] Cela pourrait être traduit par « prendre et placer ».

sont libres de se placer où ils l'entendent. Ils peuvent alors profiter d'un plus grand confort en évitant de se tenir très près de leur partenaire. Une autre raison est que les utilisateurs n'avaient pas une position particulière à adopter pour leurs poignets en début de manipulation, d'où, là encore, un plus grand confort. Enfin, il reste une raison plus pratique. L'usage direct des positions pouvait aboutir à la situation où les utilisateurs étaient comme en train de vouloir faire traverser le sol à l'objet virtuel manipulé, lorsque la manipulation démarrait. Des instabilités de l'objet apparaissaient alors.

Les positions et orientations de l'objet manipulé sont notées p_{obj} et q_{obj}, où p est une position dans un espace en 3D et q est un quaternion. Une position et une orientation fournies par un utilisateur n sont obtenues périodiquement et notées p_n et q_n. Nous obtenons :

$$p_{obj} = \frac{\vec{t_1} + \vec{t_2}}{2} + p_{obj} \qquad \text{avec } \vec{t_n} = p_n - p_{n,prev} \tag{5.1}$$

$$q_{obj} = slerp(q_1, q_2, \frac{1}{2}) \qquad \text{avec } q_n = (q_n \cdot q_{n,prev}^{-1}) \cdot q_{obj} \tag{5.2}$$

La fonction « slerp » réalise une interpolation linéaire sphérique[2] entre deux quaternions [Sho85].

En pratique, si les deux utilisateurs effectuent des mouvements opposés, l'objet virtuel reste quasiment stationnaire. Si l'un des deux utilisateurs reste inactif, l'autre devra exagérer ses mouvements pour compenser. Si les deux utilisateurs se coordonnent, comme sur la figure 5.7, alors l'objet virtuel sera déplacé avec fluidité.

FIG. 5.7 – Deux utilisateurs manipulent un capot virtuel de voiture par la technique de la moyenne. Les deux utilisateurs adoptent une posture semblable pour faciliter leurs actions.

[2]En anglais, « Spherical LinEaR interPolation ».

5.3.1.2 La séparation des degrés de liberté

Le principe de la technique séparant les degrés de liberté est de distribuer les degrés de liberté associés à l'objet à manipuler aux utilisateurs. Un utilisateur ne gère que les translations de l'objet virtuel manipulé tandis que l'autre utilisateur ne contrôle que les rotations. Cette technique est décrite dans [RSJ02], par exemple.

Afin de fournir un comportement homogène aux utilisateurs avec la technique de la moyenne, l'implémentation de cette technique travaille sur les mouvements des utilisateurs (cf. figure 5.8). Les raisons de ce choix sont les mêmes que pour la technique de la moyenne. D'ailleurs, des instabilités du capot apparaissent bien plus souvent avec cette technique, en comparaison de la moyenne, dès que les utilisateurs prennent le contrôle. En effet, la moyenne, par nature, atténue les mouvements involontaires des utilisateurs ; avec la Séparation, chaque utilisateur est en prise directe avec certains degrés de liberté de l'objet manipulé.

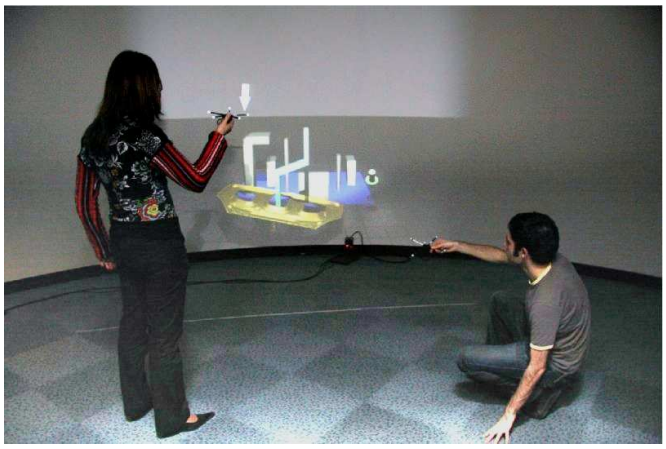


Fig. 5.8 – Deux utilisateurs manipulent un capot virtuel de voiture par la technique de la séparation des degrés de liberté. L'utilisateur debout est en train de manipuler l'orientation du capot. L'autre utilisateur est accroupi parce qu'il manipule les translations de l'objet alors que ce dernier se trouve près du sol virtuel.

5.3.2 Méthode

5.3.2.1 Conditions expérimentales

Les deux utilisateurs se tenaient face à un grand écran affichant des images stéréoscopiques. Le projecteur Barco projetait des images en 1400 x 1050 sur une surface de 3 m de long par 2 m de hauteur. L'écran était entouré de 5 caméras infrarouge de la société A.R.T. effectuant des captures de mouvements à 48 Hz pendant que l'affichage fournissait une fréquence de 48 Hz pour chaque œil. La zone scrutée par les caméras infrarouge était de 4 x4 m. Les utilisateurs portaient des lunettes actives à cristaux li-

quides et partageaient le même point de vue de la scène. Le point de vue n'était donc pas soumis à la position ou à la direction de la tête d'un des utilisateurs.

Nous avons utilisé un seul PC, composé de deux CPU Intel Pentium 4 Xeon à 3,80 Hz et d'une carte NVIDIA Quadro FX4500. L'implémentation logicielle est décrite au chapitre 6. La caméra virtuelle de la scène se déplaçait automatiquement en suivant le capot virtuel. Elle ne se déplaçait que sur la profondeur de la scène et ne changeait pas d'orientation. Elle permettait aux participants de l'expérience d'avoir une bonne vue du capot et de son environnement immédiat pour le manipuler plus efficacement. Durant l'expérience, les participants n'avaient pas besoin d'effectuer plus de deux pas en avant.

Aucun son n'était produit mais des particules jaunâtres, inspirées de [SLMA06], étaient émises lorsque le capot virtuel entrait en contact avec un autre objet virtuel. L'éclairage donnait lieu à des ombres projetées sur le sol et tout objet de la scène. Les ombres aidaient les utilisateurs à mieux évaluer la profondeur.

5.3.2.2 Procédure

L'expérimentation a fait appel à 24 participants (20 hommes et 4 femmes). Leurs âges s'étalaient de 20 à 55 ans (une personne de 41 ans et une personne de 55 ans) pour une moyenne de 26,4 ans. Peu d'entre eux (environ 21%) ont déclaré avoir une expérience en réalité virtuelle. La plupart des participants étaient des étudiants en informatique, des ingénieurs informaticiens, des chercheurs ou enseignants en informatique.

La tâche à effectuer est présentée sur la figure 5.9. Lorsque la manipulation commence, les participants doivent déplacer le capot virtuel de voiture le long d'une forme en Z pour l'extraire. Cette forme oblige les participants à fréquemment orienter le capot virtuel. De cette façon, les participants doivent coordonner leurs mouvements pour translater et orienter le capot.

Une fois que le capot virtuel a été extrait de la forme en Z, les participants doivent avancer d'un pas ou deux (en marchant) pour aller déposer le capot virtuel sur son support virtuel. Ce support est composé de deux tiges qui devront être alignées avec les deux trous situés aux extrémités du capot virtuel. En outre, une forme en T est placée sur un côté du support pour forcer les utilisateurs à suivre les étapes suivantes :

1. orienter le capot virtuel pour le mettre en position quasi-verticale ;
2. aligner un trou du capot virtuel avec la tige qui est juste à côté de la forme en T ;
3. translater le capot virtuel vers le sol ;
4. en même temps, continuer à translater le capot virtuel tout en l'orientant pour le mettre à l'horizontal.

Pour chaque technique, les participants ont reçu des explications concernant son fonctionnement. Puis, les participants ont eu quelques minutes pour essayer la technique avant de passer aux tests mesurés. Les participants étaient libres de poser des questions à l'opérateur de l'expérience durant les tests uniquement.

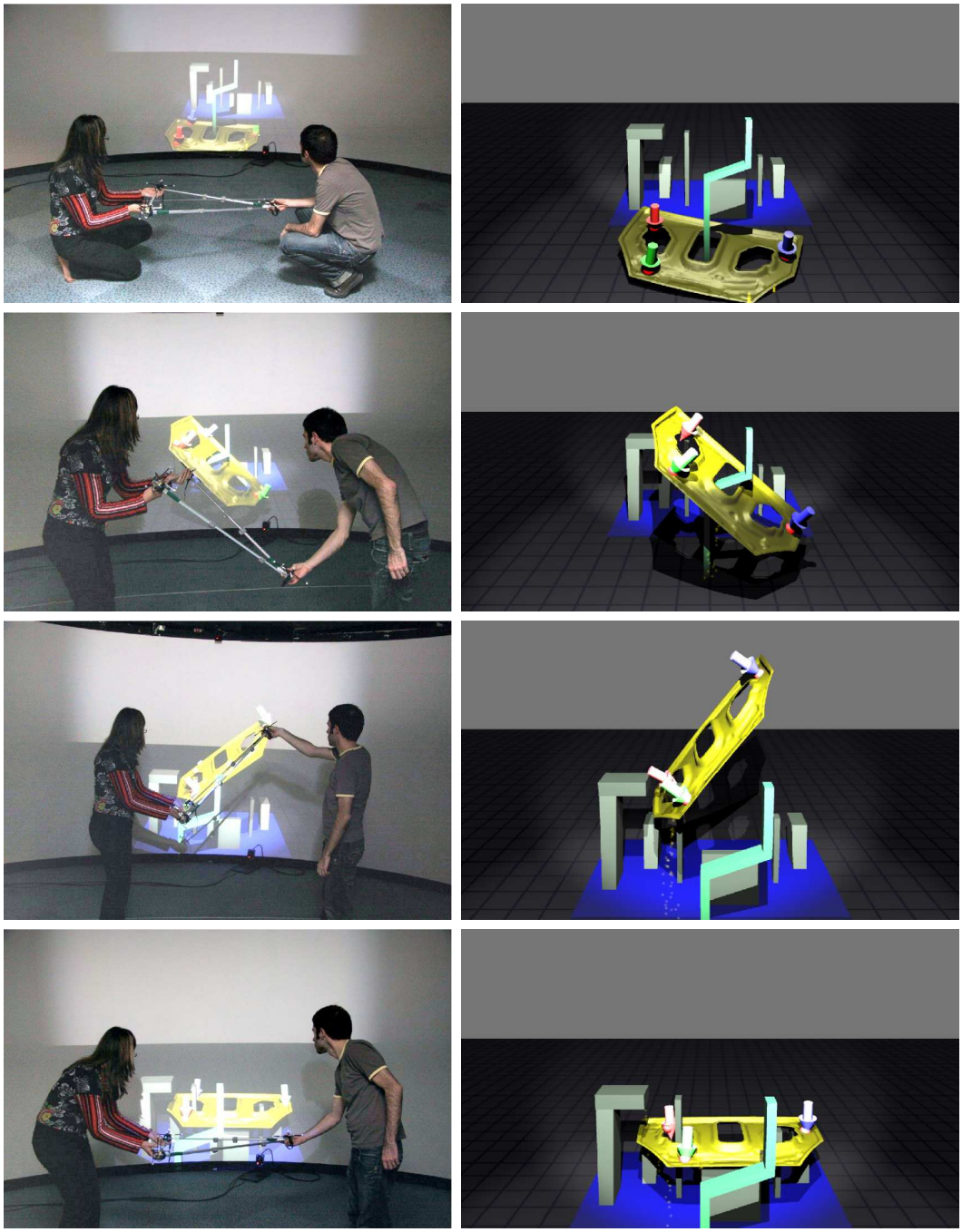


FIG. 5.9 – Étapes de la manipulation du capot par le RTD-3. Colonne de gauche : deux utilisateurs accomplissent la tâche avec le RTD-3. Colonne de droite : les mouvements correspondant au capot sont affichés. Les étapes de la manipulation sont : 1) la position initiale, 2) le capot passe le coude de la forme en Z, 3) passage entre la forme en T et une tige verticale, 4) positionnement final du capot sur le support. (Les images sont ici monoscopiques pour favoriser leur lecture.)

5.3.2.3 Plan expérimental

L'expérimentation a fait appel à 12 couples de participants. Chaque couple a testé les 3 techniques. Les participants étaient divisés en 6 groupes, ce qui correspond aux 6 ordres de présentation des 3 techniques. Deux scènes virtuelles étaient employées : l'une étant le « miroir » de l'autre[3]. Chaque couple de participants devait passer un total de 3 techniques $\times$ 2 scènes virtuelles $\times$ 2 essais $=$ 12 essais. Le premier couple d'utilisateur a suivi l'ordre : RTD (R), Moyenne (M), puis Séparation (S). Le deuxième couple a suivi l'ordre MSR. Par suite, les ordres pour les autres couples étaient : SRM, RSM, SMR et MRS. Une fois ce cycle terminé, il était recommencé jusqu'à sa fin.

À la fin de l'expérimentation, chaque utilisateur devait remplir un questionnaire comportant des évaluations subjectives (utilisant une échelle de Likert à 7 valeurs) pour chacune des 3 techniques selon les critères suivants :

1. préparation à une tâche similaire dans la réalité (**Préparation**) ;
2. réalisme de la tâche (**Réalisme**) ;
3. sensation d'immersion dans l'environnement virtuel durant la manipulation (**Immersion**) ;
4. fatigue en utilisant la technique (**Fatigue**) ;
5. impression de précision (**Précision**) ;
6. appréciation générale (**Appréciation**).

La durée totale des évaluations était d'environ 40 minutes.

5.3.2.4 Tâche dans le monde réel

Il était également proposé aux participants d'effectuer une tâche similaire dans le monde réel. La « vraie » tâche permettait aux participants de mieux visualiser la scène (virtuelle comme réelle) et de mieux comprendre les contraintes de la manipulation. La figure 5.10 illustre la tâche dans le monde réel.

Les participants devaient effectuer la tâche réelle, une seule fois, juste après avoir fini d'évaluer l'une des 3 techniques dans l'environnement virtuel et avant les 2 restantes.

5.3.2.5 Données collectées

Pour chaque technique et chaque essai, nous avons mesuré le temps d'accomplissement de la tâche et le nombre de collisions entre l'objet virtuel manipulé (*i.e.* le capot virtuel) et les autres objets virtuels peuplant la scène. Nous avons également collecté les réponses des participants au questionnaire subjectif.

5.3.2.6 Résultats

Temps d'accomplissement de la tâche. Le tableau 5.1 montre le temps moyen passé (en secondes) pour effectuer la tâche expérimentale avec chaque technique.

[3]Une symétrie orthogonale par rapport à une droite passant par le centre du support et perpendiculaire au sol de la scène virtuelle est appliquée.

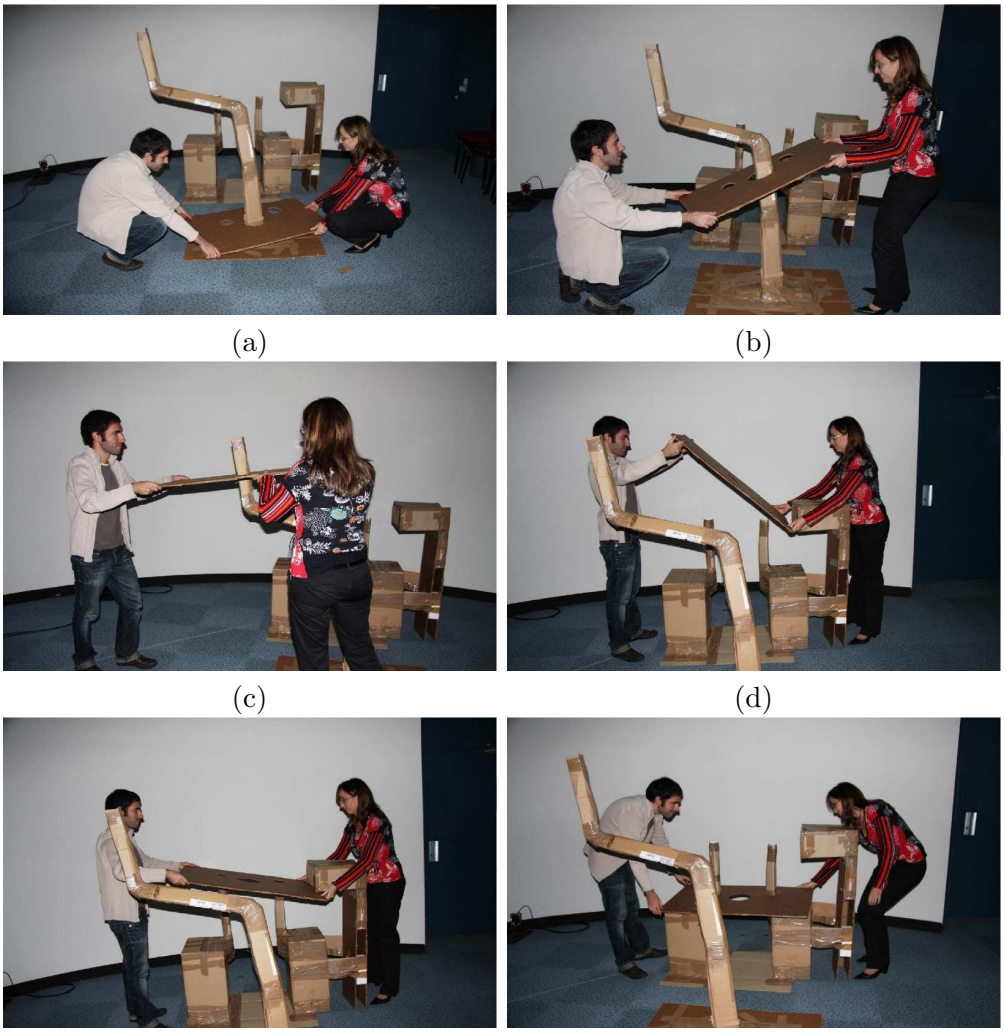


FIG. 5.10 – Tâche dans le monde réel sur une maquette en carton : la manipulation du « vrai » capot qui est aussi fait en carton.

	Moyenne	Déviation standard
RTD-3	$31,68$	$\sigma = 12,52$
Moyenne	$23,92$	$\sigma = 13,58$
Séparation	$28,17$	$\sigma = 11,48$

TAB. 5.1 – Temps d'accomplissement de la tâche (en secondes) dans l'environnement virtuel pour chaque technique.

Une ANOVA à un facteur a été appliquée sur le temps d'accomplissement de la tâche pour chacune des 3 techniques, et trois autres ANOVA à un facteur ont été effectuées pour comparer les techniques par paires. L'ANOVA globale a indiqué que le résultat pour le temps d'accomplissement de la tâche est significatif ($F(2;141) = 4,6$; $p = 0,0116$). La différence est très significative entre le RTD-3 et la Moyenne ($F(1;94) = 8,47$; $p = 0,0045$) mais pas entre le RTD-3 et la Séparation ($F(1;94) = 2,06$; $p = 0,155$), ni entre la Moyenne et la Séparation ($F(1;94) = 2,73$; $p = 0,1015$). La conclusion préliminaire est que le RTD-3 est significativement plus lent que la Moyenne.

Nombre de collisions. Le tableau 5.2 montre le nombre moyen de collisions, entre le capot virtuel et l'environnement, pour chaque technique durant la tâche.

	Moyenne	Déviation standard
RTD-3	$302,06$	$\sigma = 305,44$
Moyenne	$229,98$	$\sigma = 125,56$
Séparation	$294,81$	$\sigma = 114,64$

TAB. 5.2 – Moyennes du nombre de collisions.

À nouveau, une ANOVA à un facteur a été appliquée, cette fois sur le nombre de collisions de la tâche, pour chacune des 3 techniques, et trois autres ANOVA à un facteur ont été effectuées pour comparer les techniques par paires. L'ANOVA globale a indiqué que la différence pour le nombre de collisions n'était pas significative ($F(2;141) = 1,86$; $p = 0,16$). Cette différence n'est pas significative entre le RTD-3 et la Moyenne ($F(1;94) = 2,29$; $p = 0,1338$), même chose entre le RTD-3 et la Séparation ($F(1;94) = 0,02$; $p = 0,878$), la différence n'était significative qu'entre la Moyenne et la Séparation ($F(1;94) = 6,98$; $p = 0,0097$). Les conclusions préliminaires à propos du nombre de collisions sont que :

1. nous ne pouvons pas affirmer qu'il y a une différence significative entre le RTD-3 et la Séparation ;
2. nous ne pouvons pas affirmer que la Moyenne mène à significativement moins de collisions que le RTD-3 et la Séparation.

	RTD-3	Moyenne	Séparation
Préparation	$5,83$ ($\sigma = 1,37$)	$4,54$ ($\sigma = 1,32$)	$4,00$ ($\sigma = 1,72$)
Réalisme	$5,88$ ($\sigma = 0,99$)	$4,42$ ($\sigma = 0,88$)	$3,63$ ($\sigma = 1,54$)
Immersion	$5,71$ ($\sigma = 1,12$)	$4,88$ ($\sigma = 0,85$)	$4,54$ ($\sigma = 0,83$)
Fatigue	$4,79$ ($\sigma = 1,64$)	$4,88$ ($\sigma = 1,45$)	$5,13$ ($\sigma = 1,68$)
Précision	$4,83$ ($\sigma = 1,52$)	$4,54$ ($\sigma = 1,32$)	$4,96$ ($\sigma = 1,27$)
Appréciation	$4,96$ ($\sigma = 1,49$)	$5,04$ ($\sigma = 1,57$)	$5,08$ ($\sigma = 1,47$)

TAB. 5.3 – Moyennes et déviations standards pour les évaluations subjectives des techniques sur une échelle de Likert à 7 valeurs.

5.3.2.7 Évaluations subjectives

Le tableau 5.3 montre les résultats du questionnaire subjectif. Pour chaque critère, nous avions 24 valeurs puisque nous avions 12 paires d'utilisateurs.

Pour les 6 critères, nous avons effectué une ANOVA à un facteur pour comparer les 3 techniques, puis 3 autres ANOVA à un facteur pour comparer des paires de techniques.

Préparation à la vraie tâche. L'ANOVA globale a indiqué que la différence entre les 3 techniques était très significative ($F(2;69) = 9,71$; $p = 0,0002$). Cette différence était très significative entre le RTD-3 et la Moyenne ($F(1;46) = 11,06$; $p = 0,0017$), très significative entre le RTD-3 et la Séparation ($F(1;46) = 16,66$; $p = 0,0002$), et non significative entre la Moyenne et la Séparation ($F(1;46) = 1,5$; $p = 0,227$). Donc, pour préparer des personnes à une vraie tâche, les résultats suggèrent que le RTD-3 est perçu comme meilleur que la Moyenne et la Séparation, et que la Moyenne n'est pas significativement perçue comme meilleure que la Séparation.

Réalisme. L'ANOVA globale a indiqué que la différence entre les 3 techniques était très significative ($F(2;69) = 22,45$; $p < 0,00001$). Cette différence était très significative entre le RTD-3 et la Moyenne ($F(1;46) = 29,02$; $p < 0,00001$), très significative entre le RTD-3 et la Séparation ($F(1;46) = 35,71$; $p < 0,00001$), et significative entre la Moyenne et la Séparation ($F(1;46) = 4,71$; $p = 0,035$). Pour notre tâche, les résultats suggèrent que le RTD-3 est perçu comme plus réaliste que la Moyenne et la Séparation, et que la Moyenne est perçue comme plus réaliste que la Séparation.

Immersion. L'ANOVA globale a indiqué que la différence entre les 3 techniques était très significative ($F(2;69) = 9,72$; $p = 0,0002$). Cette différence était plutôt significative entre le RTD-3 et la Moyenne ($F(1;46) = 8,41$; $p = 0,0057$), très significative entre le RTD et la Séparation ($F(1;46) = 16,73$; $p = 0,0002$), et non significative entre la Moyenne et la Séparation ($F(1;46) = 1,88$; $p = 0,177$). Donc, pour notre tâche, les résultats suggèrent que le RTD-3 est perçu comme plus immersif que la Moyenne et la Séparation, et que la Moyenne n'est pas perçue comme immergeant vraiment mieux que la Séparation.

Fatigue. L'ANOVA globale a indiqué que la différence entre les 3 techniques n'était pas significative ($F(2;69) = 0,03$; $p = 0,85$). Il n'y avait pas de différence entre le RTD-3 et la Moyenne ($F(1;46) = 0,03$; $p = 0,85$), la différence n'était pas significative entre le RTD-3 et la Séparation ($F(1;46) = 0,48$; $p = 0,49$), ni entre la Moyenne et la Séparation ($F(1;46) = 0,48$; $p = 0,49$). Donc, pour notre tâche, les résultats ne peuvent montrer une différence significative en ce qui concerne la fatigue perçue.

Précision. L'ANOVA globale a indiqué que la différence entre les 3 techniques n'était pas significative ($F(2;69) = 0,58$; $p = 0,5619$). La différence n'était pas significative entre le RTD-3 et la Moyenne ($F(1;46) = 0,5$; $p = 0,48$), non significative entre le RTD-3 et la Séparation ($F(1;46) = 0,1$; $p = 0,759$), non significative entre la Moyenne et le Séparation ($F(1;46) = 1,25$; $p = 0,27$). Donc, pour notre tâche, les résultats ne peuvent pas montrer que la précision perçue est significativement différente entre les 3 techniques.

Appréciation. L'ANOVA globale a indiqué que la différence entre les 3 techniques n'était pas significative ($F(2;69) = 0,04$; $p = 0,9584$). Il n'y avait pas de différence entre le RTD-3 et la Moyenne ($F(1,46) = 0.04$, $p = 0.85$), pas de différence significative entre le RTD-3 et la Séparation ($F(1;46) = 0,09$; $p = 0,77$), pas de différence significative entre la Moyenne et la Séparation ($F(1;46) = 0,01$; $p = 0,925$). Donc, pour notre tâche, les résultats ne peuvent pas montrer que l'appréciation des utilisateurs est significativement différente entre les 3 techniques.

5.3.2.8 Discussion

Notre évaluation avait pour but de tester le potentiel du RTD-3 et de le comparer avec des techniques classiques d'interactions collaboratives dans des environnements virtuels. En nous appuyant sur nos résultats, nous pouvons insister sur le fait que le questionnaire subjectif est fortement en faveur du RTD-3. Ces résultats confortent ceux de Salzmann *et al.* [SJF09] où les participants tendent à préférer les interactions collaboratives effectuées au moyen d'une interface tangible plutôt que celles purement virtuelles (*i.e.* sans une interface tangible).

La plupart des participants a perçu le RTD-3 comme étant une technique réaliste. Un participant a écrit : « la stratégie dans le vrai monde était proche de celle avec le triangle », dans le questionnaire. Un autre participant a écrit : « nous essayons d'imiter ce que nous faisons dans le vrai monde ». En fait, plusieurs participants ont expliqué que le comportement du RTD-3 est proche de ce qu'ils peuvent trouver dans le monde réel. Les participants se sont également sentis bien immergés dans le monde virtuel avec le RTD-3. Un participant a écrit : « on ressent vraiment bien l'espace pris par le capot ». Plus d'immersion et un meilleur réalisme ont probablement conduit les utilisateurs à avoir l'impression d'être mieux préparés à la vraie tâche grâce au RTD-3. Un participant a notamment écrit : « l'avantage, c'est la similarité avec le vrai monde ».

Cependant, malgré les préférences subjectives des participants, l'utilisation du RTD-3 conduit à un temps d'accomplissement de la tâche qui est plus long que pour la

Moyenne. Ce résultat diffère de celui de Salzmann *et al.* [SJF09] qui ont trouvé qu'une technique d'interaction collaborative purement virtuelle (utilisant la Moyenne) prend plus de temps pour effectuer une tâche qu'une technique basée sur une interface tangible. Ce résultat pourrait être expliqué par le choix d'implémentation de la Moyenne. L'implémentation de [SJF09] moyenne les positions fournies par les participants alors que notre implémentation moyenne les mouvements des participants. Peut-être que notre implémentation permet aux participants d'adopter une posture plus confortable puisqu'ils ne sont pas contraints de rester proches l'un de l'autre. Cependant, Salzmann *et al.* ont également constaté que certaines paires de participants réussissaient à atteindre des temps comparables à ceux obtenus par l'usage d'une interface tangible. Certains commentaires de nos participants pourraient expliquer pourquoi le RTD-3 peut conduire à des temps de manipulation moins bons que pour la Moyenne. Un participant a écrit à propos du RTD-3 : « ça ressemble à la réalité donc ça implique plus d'efforts ». Le RTD-3 requiert probablement plus de mouvements physiques que les autres techniques. Des mouvements plus complexes et plus coordonnés pourraient avoir un impact sur le temps de manipulation. Des commentaires d'autres participants sont plus contrastés concernant la fatigue perçue. Pour certains participants, le RTD-3 rend l'interaction plus fatigante à cause des manipulations physiques : « on se bat pour le déplacement, parfois l'autre résiste ! » ou « fatigant si l'autre ne comprend pas ce que je veux faire ». Pour d'autres, la manipulation est plus facile et entraîne moins de fatigue : « les mouvements du partenaire sont bien ressentis, donc c'est facile de se coordonner ». Les commentaires indiquant des conflits entre les actions des deux interacteurs pourraient aussi expliquer pourquoi le nombre de collisions peut devenir élevé avec le RTD-3.

5.4 Deux interfaces tangibles reconfigurables à 4 points

L'implémentation logicielle employée pour le triangle reconfigurable, qui est décrite au chapitre 6, permet d'utiliser plus de 3 points de manipulation. Toutefois, les mouvements obtenus selon 6 degrés de liberté sont naturels si une condition est respectée : la structure supportant ces points ne doit pas changer de forme. Si les utilisateurs modifient les positions des points au cours d'une manipulation, les axes de rotation deviennent difficiles à prévoir (ceci provient de l'implémentation des contraintes employée dans Bullet). C'est cette limitation qui nous empêchait d'employer 4 points de manipulation avec uniquement des mains virtuelles au chapitre 4. L'utilisation d'une interface tangible rigide fait disparaître cette limitation.

Une interface tangible reconfigurable offrant 4 points de manipulation est nommée *RTD-4*. Deux versions ont été développées : l'une est plane, l'autre non-plane.

5.4.1 Une version plane

Une interface plane est l'extension la plus immédiate de l'interface triangulaire pour une version à 4 points. Le RTD-4 utilise des bras rigides télescopiques (du même type que ceux employés pour l'interface triangulaire) reliés entre eux par des pivots. Un bras

supplémentaire est placé sur une diagonale pour pouvoir fixer la forme de l'ensemble. Si ce bras en diagonale était absent, le RTD-4 constituerait un système sous-contraint au niveau des degrés de liberté de rotation des quatre pivots. Dans ce cas, des efforts exercés par les utilisateurs sur l'interface produiraient des rotations autour des pivots qui entraîneraient des déformations involontaires de l'interface pour les utilisateurs. Les formes pouvant être obtenues par le RTD-4 sont nombreuses : carré, rectangle, parallélogramme, trapèze ou quadrilatère d'une façon plus générale. La figure 5.12 montre diverses configurations possibles. Une des configurations est proche d'un triangle, ce qui peut permettre de l'employer d'une façon similaire à l'interface triangulaire. La figure 5.11 illustre une manipulation collaborative entre deux utilisateurs grâce au RTD-4 où chacun utilise ses deux mains.

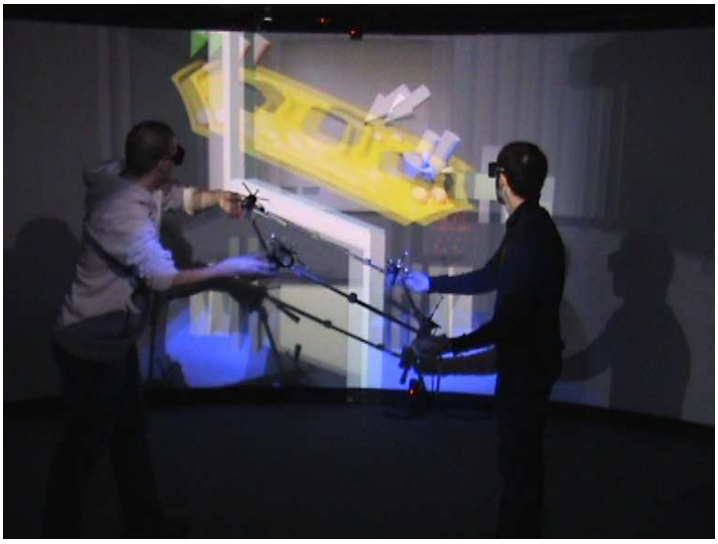


FIG. 5.11 – Manipulation d'un capot virtuel grâce au RTD-4.

5.4.2 Une version non-plane

Afin de mieux adapter la forme du maillage constitué par l'interface tangible reconfigurable, une version non-plane du RTD-4 a été créée : les pivots reliant les branches ont été remplacés par des articulations. En plus des formes proposées par la version plane, cette version non-plane permet d'obtenir des tétraèdres. Différentes configurations possibles sont illustrées par la figure 5.13.

5.5 Conclusion

Nous avons proposé un nouveau concept d'Interface Tangible Reconfigurable : une interface physique universelle qui peut correspondre à beaucoup de formes d'objets virtuels pour des manipulations virtuelles collaboratives. Cette interface repose sur l'usage

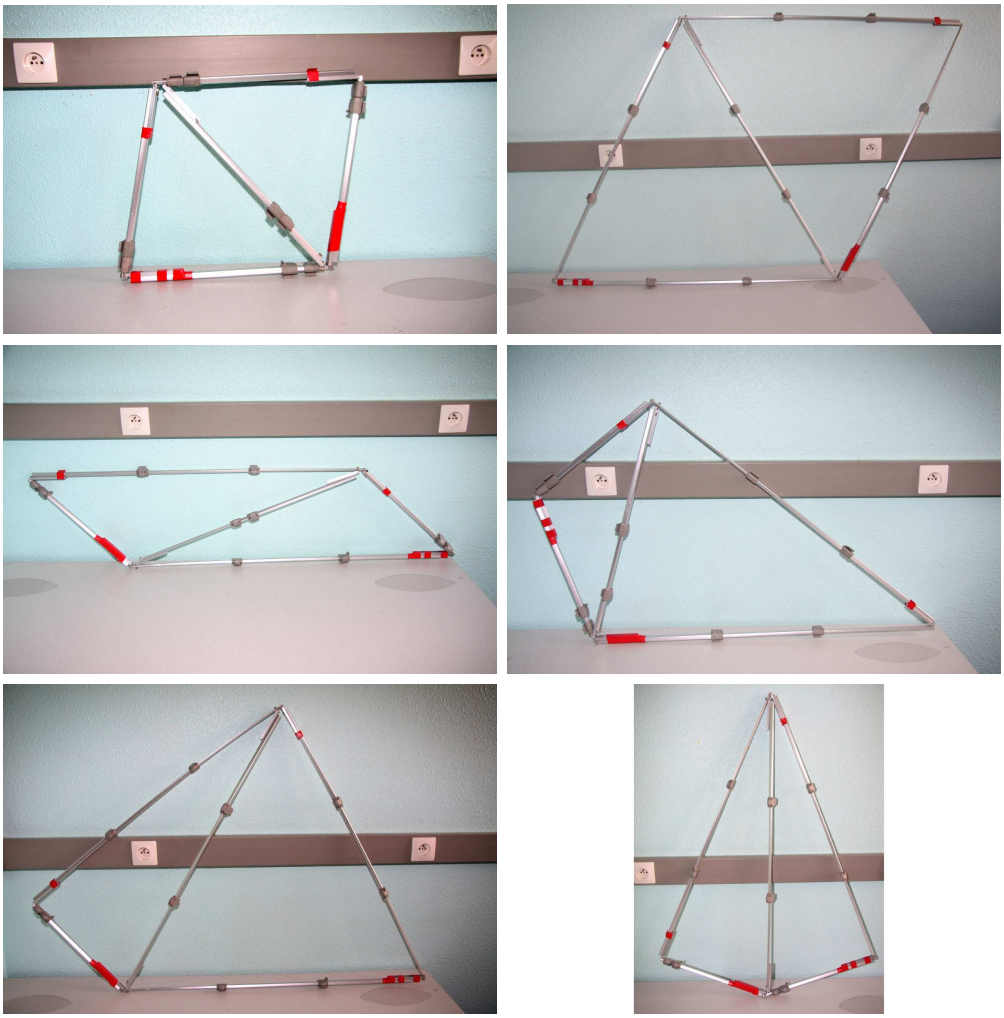


FIG. 5.12 – Différentes configurations du RTD-4 plan. Les dimensions ainsi que l'allure générale varient. En particulier, il est possible d'obtenir quasiment un triangle.

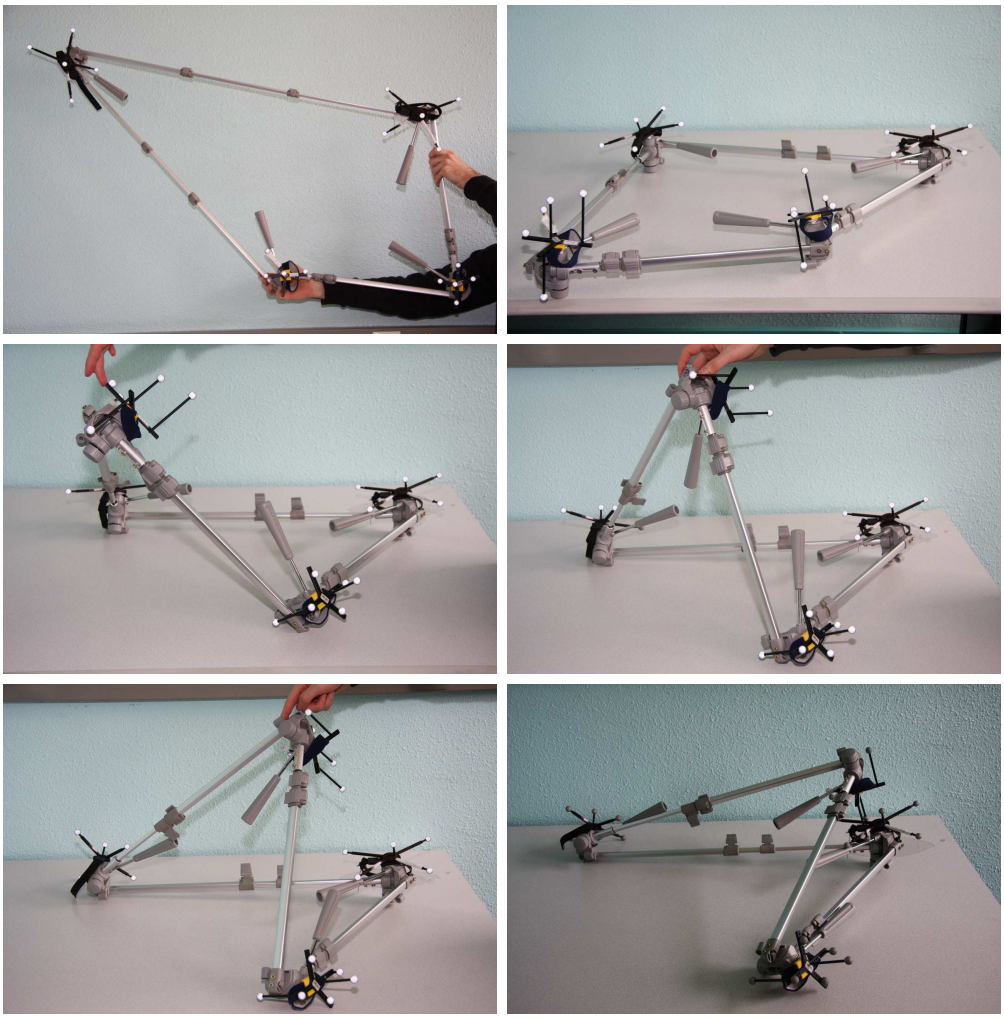


FIG. 5.13 – Différentes configurations du RTD-4 non-plan. Les dimensions ainsi que l'allure générale varient. La forme en 3D de cette interface lui permet de mieux correspondre à des objets virtuels ayant un volume.

de points de manipulation reliés entre eux par une structure rigide reconfigurable. Le maillage obtenu par ces points esquisse la forme de l'objet virtuel manipulé. Pour mieux faire correspondre ce maillage à l'objet à manipuler, les branches peuvent être étirées ou raccourcies par les utilisateurs.

Nous avons conduit une expérimentation visant à comparer l'interface tangible reconfigurable triangulaire (RTD-3) avec des techniques d'interaction collaboratives classiques : la Moyenne et la Séparation. Les résultats montrent que le RTD-3 fournit une meilleure sensation d'immersion et de réalisme. Selon les utilisateurs, le RTD-3 fournit un meilleur transfert de connaissances du virtuel vers le réel. Cependant, le RTD-3 conduit à des temps de manipulation plus longs que la Moyenne, probablement dus à plus de mouvements des corps des utilisateurs.

Globalement, nos résultats suggèrent que le RTD-3 est une bon candidat à l'entraînement de personnes accomplissant des tâches collaboratives à deux personnes. Le RTD-3 pourrait être utilisé dans plusieurs applications collaboratives telles que le prototypage virtuel ou l'entraînement aux procédures d'assemblage ou de maintenance dans des environnements virtuels.

Le RTD-4 permet à deux utilisateurs d'utiliser leurs deux mains pour interagir. Deux versions ont été présentées : l'une plane, l'autre non-plane. Le RTD-4 non-plan peut permettre de mieux faire correspondre le maillage physique obtenu par les points de manipulation à la forme 3D d'un objet virtuel à manipuler.

5.6 Perspectives

Comparaison du RTD-4 à des techniques classiques de manipulation collaboratives. Nous souhaiterions évaluer le RTD-4 (versions plane et non-plane) pour des manipulations collaboratives face à des techniques classiques de manipulation comme la moyenne ou la séparation des degrés de liberté.

Comparaison entre différents types de RTD. Il serait intéressant de comparer le RTD-4 au RTD-3 selon différents critères : vitesse d'exécution d'une manipulation, précision, ou encore impression de partage de la manipulation pour les utilisateurs.

Amélioration de l'impression d'immersion. Nous souhaiterions améliorer l'impression de co-localisation des utilisateurs du RTD afin qu'ils aient une meilleure impression de tenir réellement l'objet virtuel incarné par un RTD. L'usage d'HMD[4] pourrait aider en ce sens. Par ailleurs, nous pensons que le sentiment de l'utilisateur de tenir réellement l'objet virtuel pourrait être renforcé par l'usage de morceaux de l'objet réel au niveau des points de manipulation. Par exemple, si les utilisateurs doivent transporter un capot virtuel de voiture, les 4 points de manipulation d'un RTD-4 pourraient être des morceaux d'un vrai capot.

[4]En anglais, « Head-Mounted Display ».

Adaptation des RTD à différents scénarios. L'instance de RTD-3 ainsi que les deux instances de RTD-4 présentées dans ce chapitre ont d'abord été créées dans le but de déplacer des objets virtuels rigides composés d'un seul bloc à deux utilisateurs. Nous souhaiterions utiliser les RTD dans d'autres scénarios :

- **manipulation d'objets articulés.** En attrapant un bout articulé d'un objet, le mouvement est transmis au reste de l'objet virtuel par le biais de l'articulation qui relie ces deux parties ;
- **manipulation d'objets déformables.** Il s'agirait d'objets tels que des vêtements ou des flexibles employés sur des voitures ;
- **déformation d'objets.** La manipulation consisterait à modifier le maillage d'un objet virtuel. Dans ce contexte, les sommets du RTD-3 pourraient correspondre aux sommets d'une facette du maillage, ou bien un RTD pourrait permettre d'agrandir ou de rétrécir l'ensemble d'un objet virtuel ;
- **manipulation d'outils d'interaction ou de widgets de contrôle.** Les RTD pourraient servir à manipuler des outils d'interaction. Par exemple, la taille et la position d'un outil pour réaliser des plans de coupe pourraient être asservies à un RTD. Le rôle de certains points de manipulation pourrait être étendu. Des points pourraient servir d'emplacement pour afficher des menus de sélection ou de moyen pour contrôler des widgets. Ainsi, plusieurs facettes pourraient être sélectionnées pour leur appliquer une tesselation afin de modifier plus finement la forme de l'objet virtuel. Par ailleurs, une branche d'un RTD pourrait être vue comme un widget sur lequel un bouton peut glisser — cette idée est à faire correspondre avec un widget utilisé pour le défilement d'une fenêtre. Le bouton glissant serait réalisable par une sorte d'anneau dont les déplacements seraient suivis (par exemple par un système optique).

Pour exécuter ces cas de figures, il faudra peut être prévoir différentes tailles de RTD. Des représentations visuelles (ou autres) devront également aider les utilisateurs à interagir selon ces différents modes.

Ajout de points de manipulation. Le nombre de points de manipulation pourrait être porté à plus de 4 pour deux raisons. D'abord, cela permettrait à plus de deux utilisateurs d'employer leurs deux mains. Ensuite, certains points pourraient être dédiés au déplacement de l'objet tandis que d'autres serviraient à des opérations de contrôle de l'interaction comme nous l'avons déjà signalé.

Chapitre 6

Architecture du couplage avec un moteur physique

6.1 Introduction

Longtemps, la physique des objets a été une donnée ignorée des plates-formes de réalité virtuelle parce que sa gestion implique l'usage de calculs complexes. Certains environnements virtuels se contentaient de mécanismes de détection de collisions *ad-hoc* et, parfois, implémentaient leur propre système pour le calcul de la cinématique.

Depuis quelques années, de multiples bibliothèques apparaissent pour implémenter les calculs de physique du déplacement d'objets rigides ou déformables. Notamment, des industriels issus du jeu vidéo proposent des solutions qui sont assez rapides à mettre en œuvre, aux possibilités nombreuses et offrant des performances élevées. Par exemple, NVIDIA propose PhysX [Phy] et Intel est propriétaire de Havok [Hav]. Ces solutions offrent, entre autres choses, une gestion des collisions et une gestion de la physique d'objets rigides ou déformables.

Ce chapitre explique comment le moteur Bullet [Bul] a été employé pour la gestion de la physique d'objets rigides. Par ailleurs, cette réalisation sert de socle technique pour la manipulation à trois mains d'objets virtuels, présentée au chapitre 4. Le développement de la bibliothèque Bullet est mené par Sony Computer Entertainment US R&D. Cette bibliothèque est réalisée sous licence Zlib[1]. Elle offre notamment une gestion des objets rigides ou déformables ainsi que la gestion de collisions. Globalement, son API pour la physique partage beaucoup de notions communes avec celles de PhysX et de Havok. Ainsi, l'architecture logicielle exposée ici pourrait être portée rapidement sur ces plates-formes.

[1] Au même titre que la BSD, il s'agit d'une licence *très peu* contraignante en ce qui concerne la réutilisation et la modification du code au sein d'un code propriétaire.

6.2 Contraintes de développement

Commençons par signaler que notre architecture cherche à ne pas reposer trop fortement sur un moteur physique particulier afin de pouvoir changer celui-ci si besoin. Pour atteindre cet objectif, nous avons cherché à identifier les concepts de base d'un moteur physique et à les abstraire *via* des classes abstraites servant de fondation qui sont ensuite dérivées en des classes concrètes d'implémentation qui, elles, sont liées à Bullet.

6.2.1 Bullet vu comme un objet de simulation

Dans OpenMASK, le système de visualisation est contenu dans un objet de simulation. Cela a permis d'employer des méthodes génériques pour communiquer avec lui : événements et mécanismes d'entrées/sorties. Dans le même but, le moteur physique apparaît comme un objet de simulation quelconque. Lors d'une session distribuée, il peut ainsi être placé sur une machine où la charge du processeur est faible, ou encore qui se situe à une place qui n'avantage aucun site en particulier afin d'homogénéiser les performances.

Un objet de simulation OpenMASK contient la méthode `computeParameters` qui est appelée périodiquement pour faire évoluer l'état de l'objet. Par conséquent, cette méthode devra être exécutée à une haute fréquence pour fournir une simulation physique réaliste. Dans le cas contraire, les collisions pourraient être mal gérées et donner lieu à de nombreuses interpénétrations entre objets rigides.

À noter que Bullet essaie d'estomper les variations sur le temps écoulé entre deux pas d'exécution de la simulation physique par des mécanismes qui lui sont internes (par exemple, des interpolations de positions sont effectuées). Cette propriété permet à Bullet de fournir une simulation globalement stable malgré de légères variations de ce temps qui peuvent apparaître si des calculs sont trop longs.

6.2.2 Types d'objets physiques

Les objets de l'environnement virtuel doivent être associés à des objets physiques, c'est-à-dire des objets manipulés par le moteur physique, afin de leur donner des propriétés physiques. Bullet considère trois sortes d'objets physiques.

Objet statique. Ce type d'objet ne peut pas être déplacé mais sa surface participe aux calculs de collisions.

Objet cinématique. Ce type d'objet peut être déplacé en modifiant directement sa position. Il participe lui aussi aux calculs de collisions.

Objet dynamique. Ce type d'objet ne peut être déplaçable que par des contraintes ou des forces qui lui seraient appliquées (à la suite d'une collision avec un objet physique, par exemple). Contrairement à un objet cinématique dont la position ne dépend que de

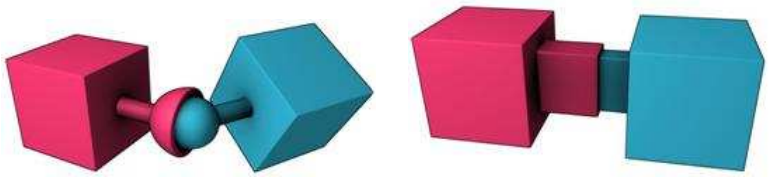
celle qu'on lui attribue, celle d'un objet dynamique évolue selon les lois de la mécanique classique : un objet non-contraint tombe parce qu'il est soumis à la gravité.

Chacun de ces objets physiques peut disposer de différentes propriétés. Par exemple, les objets dynamiques disposent d'une masse et d'une inertie. De plus, chacun voit sa forme géométrique approximée pour réduire les temps de calculs des collisions : par des boîtes, des sphères, des capsules, des maillages simples, etc.

6.2.3 Types de contraintes physiques

Dans un environnement virtuel, nous souhaitons voir des objets tomber au sol lorsqu'ils sont poussés en dehors d'une surface (ex : une table) ou lorsqu'un interacteur les laisse tomber. Dans les deux cas, nous allons passer par des objets dynamiques sur lesquels des contraintes seront appliquées pour les déplacer ou les maintenir assemblés. Les contraintes des moteurs physiques peuvent être de plusieurs types : articulations[2], couple, contact/friction, etc. Pour nous, le terme « contrainte » s'est restreint aux contraintes d'articulation.

Nous nous intéressons tout particulièrement à l'ajout dynamique de contraintes d'articulations entre un outil d'interaction et un objet interactif. Voyons quelques contraintes que nous employons.



(a) (b)

FIG. 6.1 – (a) Illustration d'une contrainte point à point (ou rotule) entre deux objets. (b) Une contrainte de type glissière.

Six degrés de liberté. Les deux objets contraignent mutuellement leurs mouvements selon six degrés de liberté. Déplacer l'un de ces deux objets entraîne le même déplacement de l'autre s'ils sont tous deux dynamiques. Il suffit alors qu'un objet soit, à la fois, un outil interacteur et un objet cinématique pour qu'il puisse déplacer et orienter l'objet interactif, qui est un objet dynamique. En effet, un objet dynamique ne peut déplacer un objet cinématique.

Point à point (ou rotule). Les deux objets ont une liaison de type rotule (cf. figure 6.1 (a)). De la même façon que pour une contrainte agissant sur les six de degrés de liberté, l'un des deux objets est, à la fois, un outil interacteur et un objet cinématique.

[2]En anglais, « joint ».

Lorsque l'interacteur déplace son outil, l'objet interactif manipulé se balance alors par rapport à lui.

Glissière. Les deux objets peuvent glisser l'un par rapport à l'autre le long d'une glissière (cf. figure 6.1 (b)). À nouveau, l'un des deux objets est, à la fois, un outil interacteur et un objet cinématique.

Notons que même si la quantité des types de contraintes que nous avons employés est faible, Bullet offre un plus large éventail de contraintes fondamentales, telle la charnière, et d'autres très spécialisées, telle la « ragdoll[3] » qui est, en fait, un ensemble de contraintes. Par ailleurs, il est possible pour un développeur de créer de nouveaux types de contraintes en héritant de classes fournies par Bullet.

6.2.4 Création dynamique d'objets physiques et de contraintes

La plupart des objets physiques sont à créer lors du chargement de l'environnement virtuel. À chaque objet interactif va être associé un objet physique. De plus, des objets physiques statiques seront là uniquement pour transformer le décor en un acteur de la simulation physique (*i.e.* les objets physiques dynamiques ne pourront le traverser). Toutefois, la création dynamique d'objets physiques peut s'avérer nécessaire, par exemple, pour un objet qui lancerait des projectiles créés au fil du temps.

En ce qui concerne la création dynamique de contraintes, elle est nécessaire pour rendre possible des interactions de déplacement avec des objets dynamiques et également pour lier entre eux des objets physiques venant d'être créés.

La création d'objets physiques, tout comme la création de contraintes, passe par l'envoi d'événements génériques dont nous exposerons les détails par la suite.

6.3 Architecture logicielle retenue

L'architecture retenue reprend des principes employés par l'architecture de la couche de visualisation d'OpenMASK. Pour la visualisation, Ogre3D [Ogr] permet de peupler dynamiquement la scène virtuelle, sur réception d'évènements, par des objets visuels et leurs animateurs. Pour la physique, Bullet doit permettre de peupler la scène virtuelle par des objets physiques et des contraintes entre ces objets.

Un diagramme de classes de l'architecture retenue est présenté sur la figure 6.2 où on peut voir que la physique est gérée au moyen d'un objet de simulation OpenMASK. La classe `PhysBase` est responsable de la création et de la destruction dynamique des objets physiques et des contraintes. À ce niveau hiérarchique, aucune ligne de code n'a de rapport avec la bibliothèque Bullet et se trouve donc être générique. Des « callbacks » réagissent à la réception d'événements de création ou de destruction et utilisent le patron de conception Fabrique [GHJV94] pour instancier un objet du type attendu. Par

[3]En français, ce terme pourrait se traduire par « poupée de chiffon ». Les « ragdolls » sont souvent employées, dans des jeux vidéos par exemple, pour simuler le comportement d'un corps humain projeté suite à un choc violent.

exemple, un événement de type `AddPhysicalObject` valué par la chaîne de caractères « MaBalle » va créer un objet de type `MaBalle`. Pour que cela puisse fonctionner, l'utilisateur aura eu à créer une classe `MaBalle` héritant de la classe `BulletRigidObject` (cf. figure 6.3), par exemple, pour obtenir un objet au comportement proche de celui d'une balle réelle.

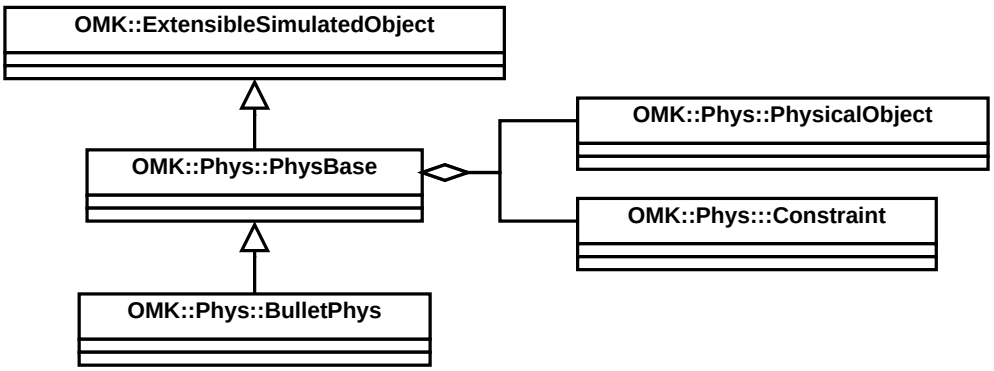


FIG. 6.2 – Diagramme de classes pour la gestion de la physique en se basant sur Bullet.

La classe `BulletRigidObject` peut représenter un objet statique, ou bien cinématique, ou bien dynamique. Cette possibilité est offerte par Bullet qui effectue une différence entre ces trois types d'objets en se limitant à deux aspects : la masse (un objet statique et un objet cinématique ont une masse nulle) et des descripteurs de l'objet (un paramètre particulier de Bullet est utilisé pour lui signaler qu'un objet est cinématique et non-statique). En pratique, une telle versatilité permet généralement d'éviter de créer des classes aussi spécifiques que `MaBalle` : seuls les paramètres de configuration ont besoin d'être correctement réglés.

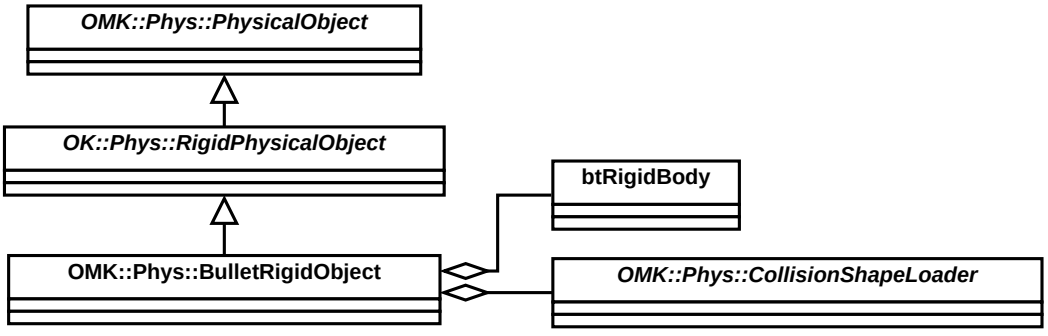


FIG. 6.3 – Diagramme de classes pour les objets physiques. La classe BulletRigidObject implémente tout type d'objet : statique, cinématique ou dynamique grâce à l'usage d'un pointeur sur un objet physique Bullet. Elle emploie également un chargeur de formes géométriques pour approximer sa géométrie.

Selon la nature d'un objet physique dans Bullet, l'objet physique d'OpenMASK associé se comporte de différentes façons. Un objet physique cinématique crée automatiquement une entrée (au sens OpenMASK) pour trouver les différentes positions à utiliser. Un objet physique statique se contente d'utiliser la première position qui lui a été fournie. Un objet physique dynamique ne lit aucune position, sauf pour se placer lors du démarrage de la simulation physique. Cependant, la position d'un objet physique dynamique peut évoluer au cours d'une simulation physique parce qu'un objet de cette sorte peut subir des collisions ou des contraintes. Afin de suivre les changements de positions d'un objet physique dynamique, tout objet physique dynamique est associé à une sortie OpenMASK qui lui est propre. Il est donc possible de connecter d'autres objets OpenMASK à ces sorties pour suivre les positions d'objets physiques dynamiques (ex : l'objet implémentant la visualisation peut placer un objet visuel correspondant à un objet physique dynamique dans la scène 3D courante).

Un objet physique n'est pas complet s'il ne dispose pas d'une approximation de sa forme pour gérer des collisions. La classe `BulletRigidObject` encapsule une instance de la classe abstraite `CollisionShapeLoader` dans ce but. Ce chargeur a pour rôle le chargement d'un ensemble de figures géométriques afin de produire la forme englobante de l'objet. Actuellement, la classe `CreateBulletCompoundCollisionShape` hérite de `CollisionShapeLoader` et permet le chargement de figures géométriques simples, décrites dans le fichier de configuration d'OpenMASK, à placer relativement les unes par rapport aux autres. La figure 6.4 illustre notamment la description d'un objet qui est une lampe.

Il est également possible de définir une géométrie pour les collisions en utilisant une autre classe héritant de `CollisionShapeLoader`. Par exemple, la classe `DynamicObject-FromMesh` permet d'obtenir un objet physique à partir du maillage d'un objet 3D créé dans Ogre3D. De cette façon, il peut devenir plus aisé d'obtenir certaines formes géométriques pour les collisions. En effet, on peut simplifier automatiquement un maillage 3D par des fonctionnalités offertes par de nombreux logiciels de modélisation géométrique. Dans le cas où cette simplification produit un maillage sans « trous » et avec un nombre de polygones faible[4], il peut alors être utilisé par notre classe.

Le diagramme de classes de la figure 6.5 montre comment sont obtenues les contraintes pour une liaison ponctuelle (point à point), une glissière ou une limitation sur les six degrés de liberté.

6.4 Intégration avec le protocole d'interaction

Nous expliquons à présent comment l'architecture que nous avons employée pour la gestion de la physique avec Bullet s'intègre au travail fourni autour du protocole d'interaction, présenté au chapitre 3. Plus précisément, nous montrons comment étendre et utiliser les extensions issues du protocole au travers de deux types d'interactions : l'interaction à trois mains virtuelles (cf. chapitre 4) et l'interaction au moyen d'un objet

[4]Les performances lors de la détection de collisions sont immédiatement affectées par un nombre trop élevé de polygones.

```
physicalLamp
{
  Class BulletRigidObject   // Type de l'objet OpenMASK à créer.
  AssociatedObject "lamp"   // Association à un objet simulé OpenMASK.
  StaticObject false        // On ne veut pas d'un objet statique.

  Position [[-3 -9 5.3]]    // La position initiale.

  // Différents paramètres pour obtenir un objet physique dynamique
  Mass 3                    // À noter la masse non nulle.
  Inertia [0.3 0.6 0.3]
  ...

  // Déclaration du type de chargeur de géométries de collision à
  // utiliser.
  CollisionShapeLoaderType "CreateBulletCompoundCollisionShape"
  Shapes
  {
    foot                    // Le pieds de la lampe.
    {
      Type box
      HalfExtends [0.75 0.17 0.75]
      LocalTransform [[0 0.14 0]]
    }
    body                    // Son mât.
    {
      Type box
      HalfExtends [0.18 3.33 0.18]
      LocalTransform [[0 3.54 0]]
    }
    top                     // L'abat-jour.
    {
      Type cone
      Radius 1.4
      Height 1.4
      LocalTransform [[0 7.58 0]]
    }
  }
}
```

FIG. 6.4 – Illustration de la description d'un objet physique dynamique (une lampe) : ses propriétés (la masse par exemple) ainsi que le chargeur à employer pour décrire sa géométrie englobante.

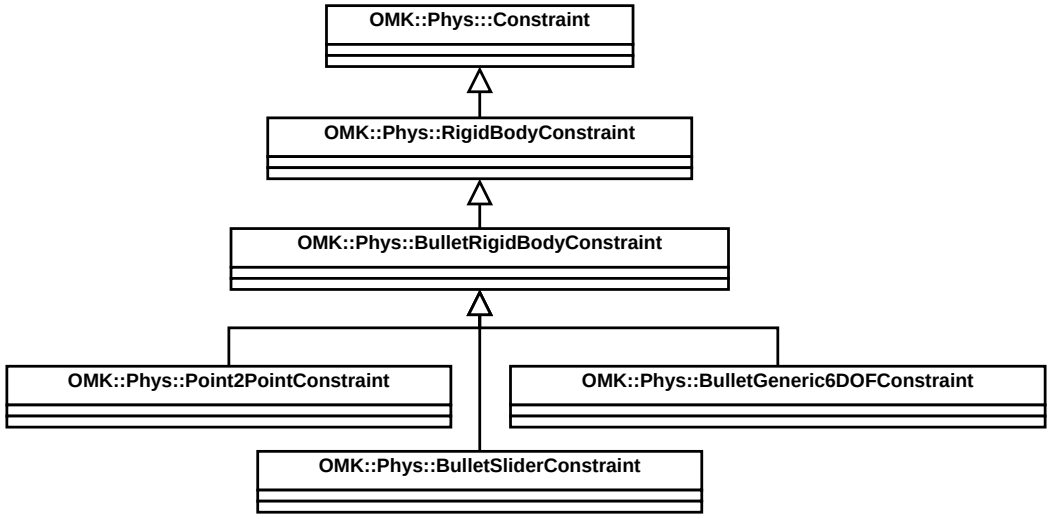


FIG. 6.5 – Diagramme de classes pour les contraintes d'articulation entre objets physiques. Seuls trois types de contraintes sont ici représentés : point à point, glissière et six degrés de liberté.

intermédiaire pour l'utilisation d'une moyenne des positions / orientations proposées ou d'une intersection de ces positions / orientations.

6.4.1 Interaction à trois mains virtuelles

Dans le cas de cette interaction, trois pointeurs vont manipuler simultanément un objet physique dynamique grâce à des contraintes point à point. Si un seul pointeur manipule, l'objet virtuel se balance au bout de celui-ci. Si deux pointeurs manipulent l'objet, il oscille autour de l'axe induit par les bouts des deux pointeurs. Avec les trois pointeurs, un plan est formé. Nous souhaitions d'abord montrer à l'utilisateur que l'utilisation de trois pointeurs virtuels peut l'aider dans ses manipulations. Pour cette raison, il est possible de saisir l'objet avec un, deux ou trois pointeurs à la fois. Au-delà de trois pointeurs, le système peut conduire à des résultats de manipulation aberrants.

Le chapitre 3 introduisait la notion de connecteur. Pour rappel, un connecteur est une pièce logicielle associée à une extension interactive d'un objet interactif. Il a pour rôle de gérer la connexion/déconnexion d'une sortie d'un outil d'interaction, ou d'un proxy-outil, à l'une des entrées de l'objet interactif. Il va également faire appel à un convertisseur pour gérer, par exemple, la fusion de données provenant simultanément de plusieurs outils d'interaction ou proxy-outils.

Une vue d'ensemble de la connexion des outils d'interaction (ici, les « mains virtuelles ») est fournie par la figure 6.6. On peut observer que les trois outils communiquent avec l'objet interactif de la même façon que lorsque la gestion de la physique n'était pas présente. Cependant, le connecteur ne va pas appeler le convertisseur : il se contente de connecter les outils pour ne pas modifier leur comportement. Les ou-

tils, quant à eux, ont chacun été connectés dès le lancement de la simulation virtuelle, au moyen du fichier de configuration d'OpenMASK, à un objet physique cinématique contenu dans l'objet de simulation implémentant Bullet. Le connecteur de l'objet interactif va envoyer des événements à Bullet pour que des contraintes point à point soient ajoutées entre un objet physique cinématique représentant une main / un pointeur virtuel et l'objet physique dynamique manipulé représentant un capot virtuel de voiture. La sortie `Position` de l'objet physique dynamique associé au capot est connectée à l'attribut `Position` de l'objet interactif représentant le capot. De cette façon, des objets peuvent venir se connecter à la sortie de l'objet interactif représentant le capot exactement comme si la gestion de la physique n'était pas présente. Il est d'ailleurs possible de voir sur la figure 6.6 que le système de visualisation est connecté à l'objet interactif représentant le capot afin de l'afficher à l'écran.

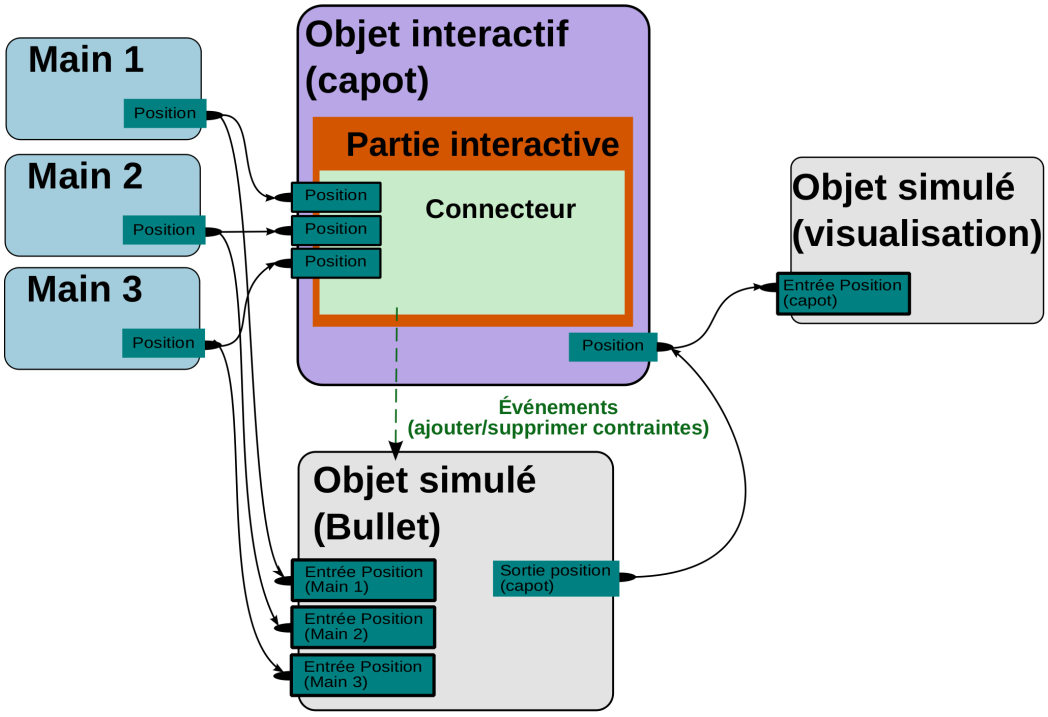


FIG. 6.6 – Deux outils manipulent la position d'un capot grâce à un objet interactif. La visualisation est connectée à la sortie de l'objet interactif représentant le capot afin de l'afficher, correctement positionné, à l'écran.

Pour expliquer comment les connecteurs ont été employés dans le cas de la manipulation à trois mains virtuelles, commençons par restreindre un connecteur à une classe ne disposant que de trois méthodes : `connect`, `updateParameter` et `disconnect`.

La connexion. La méthode `connect` effectue la connexion entre une sortie d'un proxy-outil et une entrée de l'objet interactif. Le code employé est globalement celui

utilisé lorsque nous ne souhaitons pas ajouter une contrainte. La différence est l'étape d'obtention du point d'intersection entre le rayon « lancé » par un pointeur virtuel et l'objet interactif à manipuler. Nous cherchons un tel point pour savoir où placer une contrainte par rapport à l'objet. Par exemple, le point d'intersection est la position de la rotule lorsque la contrainte rotule est employée.

Pour obtenir le point d'intersection issu du rayon lancé, il faut envoyer un message d'interrogation (en terme OpenMASK, on envoie un événement) au pointeur virtuel. Ce n'est qu'après avoir reçu une réponse de sa part, ou après échec sur non-réponse, que la manipulation de l'objet virtuel débutera. Par conséquent, un délai, dépendant de plusieurs facteurs, est introduit. Sur OpenMASK, l'envoi d'un événement n'est effectué qu'à la fin d'un pas de simulation de l'objet correspondant. La réception d'un événement est fonction de la fréquence de l'objet de simulation qui le reçoit. Donc, le délai dépend principalement du délai introduit par le réseau (si un réseau est employé) reliant le site ayant l'outil d'interaction au site ayant l'objet interactif, et du temps de traitement / fréquence des objets de simulation d'OpenMASK. Notre architecture fait donc la supposition qu'un objet interactif à « attraper » aura toujours une vitesse faible par rapport aux latences de communication. Si cette supposition n'est pas satisfaite, un outil d'interaction pourrait avoir des difficultés à attraper un objet interactif.

La mise à jour d'attributs. La méthode `updateParameter` est appelée régulièrement par l'objet interactif lui-même pour faire évoluer son état. Pour des raisons techniques, c'est également dans cette méthode que l'ajout de la contrainte est effectué. En effet, entre l'instant où la position du point d'intersection est demandée et l'instant où elle est reçue, du temps peut s'écouler. Or, une extension interactive emploie la méthode `updateParameter` dès que le connecteur correspondant est connecté. L'algorithme 1 montre que dès que le point d'intersection est connu, un événement demandant l'ajout d'une contrainte est construit puis envoyé à l'objet de simulation qui gère Bullet. Enfin, on garde en mémoire que la contrainte a été ajoutée afin d'éviter de l'ajouter à nouveau sans l'avoir supprimée au préalable.

Algorithme 1 Ajout d'une contrainte.

```
si le point d'intersection du rayon Proxy-Outil/Objet interactif est connu alors
  config ← chargerConfigurationContrainte() ;
  objetsParticipant ← {proxyOutil, objetInteractif}
  positionDuPivotDansLEspace ← pointDIntersectionRayonProxyOutilObjet
  ev ← Créer( TypeAjoutContrainte( TypeContrainte, objetsParticipants,
                                   positionDuPivotDansLEspace, config ) ) ;
  envoyerÉvénement( Bullet, ÉVÉNEMENT_AJOUT_CONTRAINTE, ev ) ;
  contraintesAppliquées.ajouter( Créer( Liaison( proxyOutil, objetInteractif ) ) ) ;
fin si
```

La déconnexion. La méthode `disconnect` revêt deux rôles : fermer la connexion entre une sortie d'un proxy-outil et une entrée de l'objet interactif, et enlever la contrainte

qui les reliait. D'une façon similaire à celle mise en œuvre dans `updateParameter`, un événement est envoyé à l'objet de simulation gérant Bullet pour demander la suppression d'une contrainte précise.

6.4.2 Interaction par un objet OpenMASK intermédiaire

Dans le chapitre 5, l'interface tangible reconfigurable a été évaluée en la comparant aux techniques de la moyenne des translations et rotations fournies, et la séparation de degrés de liberté. Pour parvenir à ce résultat, nous avons dû introduire un objet simulé intermédiaire, noté I, entre l'objet virtuel à manipuler et les interacteurs.

L'objet virtuel à manipuler est, dans ce paragraphe, le capot virtuel de voiture qui a été employé dans les chapitres 4 et 5.

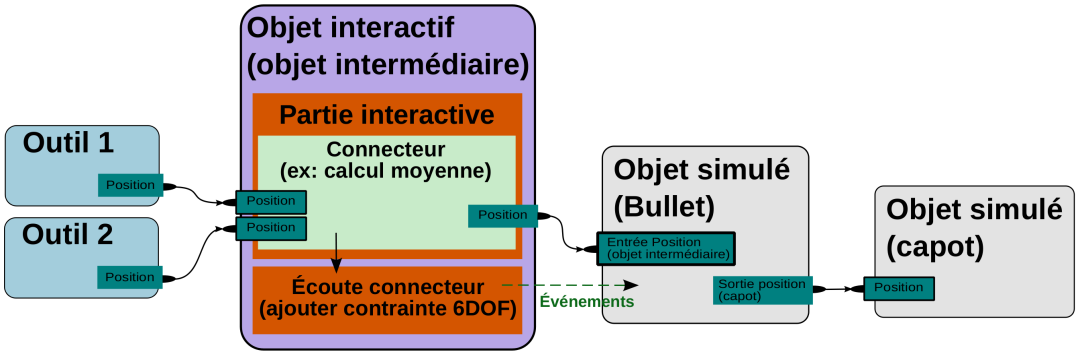


FIG. 6.7 – Deux outils manipulent un capot virtuel à travers un objet intermédiaire qui calcule des moyennes de positions et d'orientations.

La figure 6.7 montre deux outils proposant des positions à l'objet intermédiaire I. Cet intermédiaire est un objet interactif contenant un connecteur. Ce connecteur emploie un convertisseur qui effectue des moyennes de positions et d'orientations. Un outil n'interagit plus directement avec le capot mais communique systématiquement avec l'objet simulé intermédiaire I. La moyenne calculée est envoyée à l'objet simulé qui implémente la gestion de la physique grâce à Bullet. Cet objet simulé comporte un objet physique cinématique, noté P_I, associé à l'objet intermédiaire I, pour que l'objet physique puisse être déplacé aisément. De plus, un objet physique dynamique, noté P_C est également présent pour représenter le capot. Lorsque le connecteur de l'objet intermédiaire entre en fonction, il envoie à Bullet un événement pour lui demander d'ajouter une contrainte à six degrés de liberté entre P_I et P_C. Ainsi, lorsque l'objet interactif est déplacé, il déplace P_I qui à son tour déplace P_C. Enfin, un objet simulé classique comportant un attribut `Position` est relié à la sortie `Position` du capot de l'objet Bullet, ce dernier est connecté à l'objet de visualisation pour déplacer un capot à l'écran.

La contrainte entre P_I et P_C est supprimée lorsqu'il n'y a plus aucun outil en train de manipuler le capot. Par conséquent, le capot peut alors tomber au sol ou être poussé par un autre objet.

Enfin, c'est au niveau du convertisseur de I que le calcul effectué peut être changé. Outre la moyenne, la séparation des degrés de liberté et l'intersection ont été implémentées.

Séparation des degrés de liberté. Seule la translation en provenance de la main M_1 est utilisée. La main M_2 ne peut fournir que des orientations. L'identification de ces mains est réalisée par le biais du fichier de configuration d'OpenMASK. En se basant sur la position précédemment reçue, le convertisseur calcule un déplacement à appliquer à l'objet. Un traitement similaire est effectué pour la rotation à calculer.

Intersection. Dans le cas de l'intersection, la translation, selon l'axe X du repère, à appliquer au capot est la quantité de mouvement commune à celle proposée par la main M_1 et celle proposée par la main M_2. Ici, les deux mains fournissent simultanément une translation et une rotation. La quantité Q_x sur l'axe X se calcule de cette façon :

$$\delta = \begin{cases} 1 & \text{si } \vec{T_x}(M_1) \geq 0, \\ \text{-1} & \text{si } \vec{T_x}(M_1) < 0 \end{cases}$$

$$Q_x = \begin{cases} \delta \times min(|\vec{T_x}(M_1)|, |\vec{T_x}(M_2)|) & \text{si } \vec{T_x}(M_1) \times \vec{T_x}(M_2) > 0, \\ 0 & \text{sinon.} \end{cases}$$

où $\vec{T_x}(M)$ est la translation d'une main M projetée sur l'axe X. La translation est obtenue à partir de la position précédente comme pour la technique de séparation des degrés de liberté. Dans le cas où les deux translations s'effectuent dans des sens opposés, la quantité de mouvement est nulle. Les quantités Q_y et Q_z sont obtenues de façon similaire pour les axes Y et Z.

L'orientation est obtenue à partir des orientations fournies par les deux mains. Les orientations sont représentées par des quaternions. Il n'est ici plus question de calculer un angle entre deux orientations successives pour obtenir une rotation à appliquer mais d'utiliser directement l'orientation commune aux deux mains. Pour cela, deux vecteurs unité sont employés. Le premier voit sa direction modifiée par l'orientation de la main M_1. Le deuxième est orienté par M_2. Si le produit scalaire entre les deux vecteurs est proche de 1 alors les orientations fournies par les deux mains sont proches. On effectue une interpolation linéaire sphérique[5] entre les deux quaternions représentant les orientations des mains et on applique le résultat à l'objet virtuel manipulé. Cependant, avant leur orientation, les deux vecteurs unité sont égaux à $(1; 0; 0)$. Par conséquent, une rotation autour de l'axe X donnera $(1; 0; 0)$, ce qui correspond à une rotation identité. Pour traiter cette situation, deux autres couples de vecteurs unité égaux à $(0; 1; 0)$ sont eux aussi orientés par les mains M_1 et M_2. Finalement, il faut donc tester si les deux

[5]Il s'agit de « slerp » (spherical linear interpolation). Déjà employée au chapitre 5 pour la technique de la Moyenne, cette méthode interpole deux quaternions [Sho85].

couples de vecteurs unité voient leurs produits scalaires proches de 1 avant d'employer les orientations proposées.

6.5 Conclusion

L'architecture proposée offre une création aisée d'objets physiques dynamiques, cinématiques ou statiques pour peupler un environnement virtuel. Cependant, un outil présentant une interface graphique rendrait cette étape de création plus aisée encore. Du point de vue des performances, notre architecture introduit une latence au moment de la saisie d'un objet virtuel. Une fois l'objet saisi, les performances dépendent de la bibliothèque de gestion de la physique puisqu'elle est seule responsable du maintien des contraintes entre des objets physiques. Toutefois, des fortes latences sur le réseau reliant les différents sites géographiques d'une simulation pourraient entraîner des incohérences entre des positions (ex : un site envoie une position en se basant sur la valeur d'une « vieille » position) mais cela n'est plus un problème spécifique à l'architecture présentée car inhérent aux architectures réseau distribuées.

Enfin, cette architecture n'a, pour l'instant, été testée que sur des objets rigides alors que Bullet, comme d'autres moteurs physiques, gère aussi les objets déformables. L'utilisateur aurait alors la possibilité de manipuler dans l'environnement virtuel des objets tels qu'un pneu ou un flexible de voiture.

Chapitre 7

Conclusion et perspectives

Le travail de cette thèse portait sur les interactions collaboratives dans les environnements 3D immersifs et a donné lieu à des contributions se présentant en deux parties. D'abord, nous avons créé un protocole d'interaction entre outils d'interaction et objets interactifs. Puis, en nous appuyant sur ce protocole comme fondation, nous avons créé la technique d'interaction collaborative à 3 mains virtuelles et l'interface tangible reconfigurable.

Nous avons débuté ces recherches par un état de l'art des techniques d'interaction dans les environnements virtuels immersifs. Il établit une progression des techniques mono-utilisateur aux techniques multi-utilisateurs en passant par l'étape des interactions bi-manuelles. De cette étude, nous avons tiré des mécanismes généraux pour l'intégration d'actions collaboratives [RSJ02] : la séparation / répartition des degrés de liberté en fonction des interacteurs, la moyenne des actions ou encore l'intersection des actions. Nous avons remarqué que certaines techniques mono-utilisateur pouvaient être converties au multi-utilisateur avec peu d'effort : « Bent Pick Ray » pour le rayon virtuel [RHWF06] ou mains virtuelles [GMMG08, SJF09]. L'effort est à placer sur le retour d'informations aux utilisateurs, qu'il soit visuel, sonore ou même haptique. Puis, nous avons été amené à considérer la façon dont les techniques d'interaction sont décrites. Nous avons alors pu tenir compte du travail effectué sur les plates-formes de réalité virtuelle pour abstraire le matériel. Nous avons ensuite discuté des langages tels que VRML [VRM] ou X3D [X3D] qui décrivent des environnements virtuels mais avec peu de formalisme en ce qui concerne les outils d'interaction et les objets interactifs. Des extensions à X3D, comme InTML [FGH02] et CONTIGRA [DHM02], cherchent cependant une formalisation d'outils ou de techniques d'interaction. Enfin, nous avons terminé par une étude des architectures réseau des plates-formes de réalité virtuelle [DWM06a, DWM06b].

Pour présenter nos contributions, nous avons choisi de débuter par le protocole d'interaction parce que cela respecte l'ordre chronologique de cette thèse et l'ordre généralement suivi dans le développement logiciel, nous avons donc commencé par poser des bases. La technique d'interaction à 3 mains et l'interface tangible reconfigurable ont participé à la validation du protocole ainsi que l'architecture logicielle sous-jacente.

Un protocole entre outils d'interaction et objets interactifs. Nous avons d'abord souhaité formaliser les échanges ayant lieu entre un outil d'interaction et un objet interactif lors d'une interaction. L'implémentation du protocole d'interaction nous a amené à créer une architecture logicielle à base d'*extensions* qui sont des petits composants logiciels facilement extensibles et réutilisables. Les extensions ont été employées pour créer des modules de désignation / sélection d'objets virtuels, de manipulation et pour fournir des retours d'informations aux utilisateurs. Ces extensions ont aussi pour but de former un ensemble d'éléments parmi lesquels l'utilisateur d'un outil-auteur viendra piocher des briques logicielles existantes et pourra en créer des nouvelles. Le but est de faciliter la création d'environnements virtuels.

Nous avons proposé une description des propriétés interactives d'objets interactifs par le bais du langage COLLADA [COL]. Cette description est indépendante d'une plate-forme de réalité virtuelle particulière. D'autre part, cette description n'est qu'une extension du langage COLLADA, ce qui permet de lire des fichiers comportant de telles descriptions sur des applications qui ne savent pas l'interpréter sans que cela pose de problèmes (ces extensions seront simplement ignorées).

L'intérêt du protocole est aussi de tendre vers une interopérabilité entre plates-formes de réalité virtuelle. L'interopérabilité permet aux utilisateurs de conserver leur plate-forme logicielle où des développements ont pu être effectués. Pour tendre vers l'interopérabilité, le protocole s'est abstrait de langages de programmation particuliers et considère une plate-forme logicielle de réalité virtuelle à base d'objets (à prendre au sens d'objets en général dans le contexte de la réalité virtuelle, c'est-à-dire pouvant être des outils d'interaction ou des objets interactifs) communiquant entre eux *via* des flux qui circulent de sorties à entrées. Le modèle entrées / sorties suit une tendance actuelle dictée par les outils-auteurs qui offrent une programmation graphique 2D des environnements virtuels.

Une technique d'interaction collaborative à 3 mains virtuelles. Nous avons proposé une nouvelle technique d'interaction collaborative qui permet à deux ou trois personnes d'interagir par le biais de 3 mains virtuelles. Seules les positions des 3 mains virtuelles sont exploitées. En n'utilisant que 3 points non-alignés, nous obtenons un triangle et nous sommes sûrs de conserver un plan. Grâce à l'assurance de conservation d'un plan, il nous a été possible de satisfaire notre objectif de ne pas être obligé d'utiliser une interface haptique, parce qu'un tel dispositif peut être coûteux et difficile à employer. Nous réussissons également à permettre à ses utilisateurs d'effectuer des mouvements naturels.

Les tests informels, que nous avons réalisés au cours de démonstrations scientifiques ou de salons grand public, nous ont permis de constater que cette méthode est assez rapidement prise en main par ses utilisateurs. De plus, le rôle des élastiques visuels, que nous rajoutons entre une main virtuelle et le point de manipulation correspondant sur l'objet virtuel manipulé, paraît être rapidement assimilé par les utilisateurs. En effet, les utilisateurs semblent s'appuyer sur les élastiques pour détecter quand leurs actions ne sont pas réalistes.

Une interface tangible reconfigurable. Nous avons proposé le concept d'interface tangible reconfigurable qui a d'abord été créée pour aider les utilisateurs de la technique à 3 mains virtuelles à coordonner leurs mouvements. En effet, le lien physique qu'elle procure permet à ses utilisateurs de conserver leurs mains physiques, et donc leurs mains virtuelles, toujours à la même distance. Pour la personne qui utilise ses deux mains physiques, l'interface tangible aide à coordonner ses 2 membres : quand l'une monte, l'autre descend pour changer l'orientation d'un objet virtuel manipulé. Le lien physique peut être qualifié de retour pseudo-haptique puisqu'il fournit aux utilisateurs, d'une part l'impression de tenir l'objet virtuel manipulé dans leurs mains, et d'autre part la possibilité de se transférer des efforts : si un utilisateur tire vers lui l'interface tangible, l'autre utilisateur vient vers lui.

L'interface tangible est reconfigurable pour pouvoir faire en sorte que ses points de manipulation correspondent à la forme de l'objet virtuel à manipuler. Le lien physique entre deux points de saisie peut être allongé ou raccourci. Enfin, l'interface tangible reconfigurable se présente actuellement sous la forme d'un triangle parce que, d'une part, elle venait en complément de la technique à 3 mains virtuelles, et, d'autre part, 3 points constituent le strict minimum pour définir un plan.

Perspectives

Les travaux présentés dans cette thèse apportent : une première brique pour répondre au problème de l'interopérabilité entre plates-formes logicielles de réalité virtuelle, une nouvelle technique d'interaction collaborative (à 3 mains virtuelles) et une nouvelle famille d'interfaces tangibles.

À propos du protocole d'interaction et des extensions à COLLADA. La question de l'interopérabilité est difficile et nécessite un développement sur la durée. De notre côté, le fait d'employer le protocole à travers différentes situations permettra de le compléter mais il est indispensable d'arriver à fédérer des industriels et des académiques. C'est uniquement en élargissant le champ des applications de réalité virtuelle possibles que ce protocole pourra se répandre. Quant à la description des interactions par COLLADA, un traitement similaire est indispensable. Les enjeux sont importants autant pour le protocole que pour la description des interactions. Il s'agit d'obtenir des moyens qui garantissent aux usagers de la réalité virtuelle d'employer des plates-formes logicielles, non parce qu'ils y sont contraints pour communiquer avec un parc de machines utilisant une plate-forme particulière, mais parce qu'elles sont les plus adaptées à leurs besoins ; il s'agit de faciliter le changement d'une plate-forme logicielle ou d'un logiciel de réalité virtuelle. Un but serait donc une standardisation du protocole et des extensions à COLLADA, voire de la normalisation des deux.

À propos de la technique d'interaction à 3 mains virtuelles. Il nous paraît nécessaire d'évaluer cette technique d'une façon formelle, autant qualitativement que quantitativement. Par rapport à des techniques comme la moyenne des actions ou la

séparation des degrés de liberté, elle est plus longue à appréhender, sans toutefois être difficile (les essais dans des salons divers le prouvent). L'évaluation devrait s'étaler un peu sur le temps pour noter une progression dans l'usage de cette technique parce qu'on ne peut la comparer « à froid » avec une technique comme la Moyenne sous peine de voir, très probablement, une différence d'appréciation immédiate en défaveur de la technique à 3 mains.

Il serait probablement intéressant de voir dans quelle mesure l'usage d'un retour pseudo-haptique pourrait faire mieux percevoir la masse d'un objet ou, plus simplement, que cet objet virtuel est en cours de manipulation. On pourrait ainsi travailler sur le rapport entre l'amplitude d'un mouvement effectué par un utilisateur et l'amplitude du mouvement obtenu à l'écran [DLB+05] où des mouvements amples de l'utilisateur ne conduisant qu'à des mouvements d'une amplitude moyenne à l'écran lui feraient imaginer que l'objet virtuel manipulé est lourd.

L'échange de certaines contraintes rotule au niveau des points de manipulation permettrait de limiter les mouvements disponibles de l'objet manipulé. Le but serait de mettre ainsi en place un mécanisme de guidage qui pourrait être utile lors d'opérations de manipulation précises, par exemple pour des ajustements au moment de l'encastrement de deux objets virtuels.

Certains points de manipulation pourraient voir leur hauteur fixée par un seul utilisateur puis, ce même utilisateur qui utilise deux points de manipulation, pourrait se déplacer tout en voyant le point de manipulation restant le suivre. En mettant à disposition des utilisateurs un tel système de déplacement automatique de certains points de manipulation, un utilisateur seul pourrait en partie évaluer le déroulement de la même manipulation si elle était réalisée avec un partenaire.

À propos de l'interface tangible reconfigurable. La technique de manipulation à 3 mains virtuelles et l'interface tangible reconfigurable (RTD) poursuivent un même but : donner la possibilité à leurs utilisateurs de placer à leur guise leurs mains virtuelles sur un objet virtuel à manipuler. Un travail doit être accompli dans la conception de telles interfaces. Elles doivent être légères pour faciliter leur manipulation, chaque branche doit autoriser un débattement important (d'une courte à une grande longueur) et les degrés de liberté autorisés par les articulations doivent être manipulables sans sensations de frottements.

Le RTD-3 devrait être comparé au RTD-4 selon des critères tels que la vitesse de réalisation d'une tâche, ou encore le degré de partage de la manipulation ressenti par les utilisateurs. Ces évaluations devraient aussi ne pas se limiter à la manipulation d'objets rigides. En effet, les positions de points d'accroches sur des objets articulés ou déformables peuvent influer sur la facilité de manipulation de ces objets parce que leurs formes pourraient changer. Ensuite, le RTD-4 devrait aussi être comparé à des techniques classiques de manipulation comme la Moyenne ou la séparation des degrés de liberté.

Différents type de manipulations devraient être évalués. En effet, un RTD peut permettre le déplacement d'un objet virtuel. À cela, nous souhaiterions ajouter l'usage des RTD pour redimensionner des objets virtuels. De plus, les RTD pourraient servir à

modifier le maillage d'objets virtuels 3D en venant associer des points de manipulation d'un RTD à des points du maillage d'un objet considéré.

L'usage de « Head-Mounted Displays » (HMD), dont les mouvements sont suivis pour estimer la direction du regard d'un utilisateur, aiderait à co-localiser l'objet virtuel manipulé et le RTD. Il faudrait alors évaluer les différences de performances entre cette situation et celle exploitée jusqu'à présent où les utilisateurs ont à regarder un écran commun avec un seul point de vue.

Avec des HMD, peut-être pourrions-nous associer aux points de manipulation des rôles autres que le déplacement de l'objet ou la modification de ses formes. Par exemple, on pourrait associer aux points de manipulation des nouvelles sortes d'interaction, comme la définition d'un plan de coupe au moyen de points de manipulation, ou bien ils pourraient être associés à des actions de contrôle de l'application (avec des menus qui apparaissent à hauteur des points de manipulation pour changer des paramètres de l'application).

Bibliographie

[AD08] Laurent Aguerreche and Thierry Duval. Formalisation des interactions collaboratives en environnements virtuels collaboratifs. In *Actes des journées de l'association française de réalité virtuelle*, pages 103–109. AFRV, octobre 2008.

[ADA09a] Laurent Aguerreche, Thierry Duval, and Bruno Arnaldi. A description of a dialog to enable interaction between interaction tools and 3D objects in collaborative virtual environments. In *Proceedings of the Virtual Reality International Conference*, pages 63–73. Simon Richir & Akihiko Shirai, avril 2009.

[ADA09b] Laurent Aguerreche, Thierry Duval, and Bruno Arnaldi. Analyse de techniques de coopération en environnements virtuels 3D. *Techniques et sciences informatiques*, 28(6–7) :767–797, 2009.

[ADL09] Laurent Aguerreche, Thierry Duval, and Anatole Lécuyer. 3-Hand Manipulation of Virtual Objects. In *Proceedings of JVRC 2009 (Joint Virtual Reality Conference of EGVE - ICAT - EuroVR)*, pages 153–156, 2009.

[ADL10a] Laurent Aguerreche, Thierry Duval, and Anatole Lécuyer. Comparison of Three Interactive Techniques for Collaborative Manipulation of Objects in Virtual Reality. In *Proceedings of Computer Graphics International*, 2010.

[ADL10b] Laurent Aguerreche, Thierry Duval, and Anatole Lécuyer. Reconfigurable Tangible Devices for 3D Virtual Object Manipulation by Single or Multiple Users. In *VRST'10 : Proceedings of the ACM symposium on Virtual Reality Software and Technology*, 2010.

[ADLA09] Laurent Aguerreche, Thierry Duval, Anatole Lécuyer, and Bruno Arnaldi. Demonstration : 3-hand manipulation of virtual objects. In *Annex of the Proceedings of JVRC 2009 (Joint Virtual Reality Conference of EGVE - ICAT - EuroVR)*, pages 45–46, 2009.

[AFM+99] David Anderson, James L. Frankel, Joe Marks, Darren Leigh, Eddie Sullivan, Jonathan Yedidia, and Kathy Ryall. Building Virtual Structures with Physical Blocks. In *UIST '99 : Proceedings of the 12th annual ACM symposium on User Interface Software and Technology*, pages 71–72, 1999.

[AFT03] Bruno Arnaldi, Philippe Fuchs, and Jacques Tisseau. *Traité de la réalité virtuelle*, volume 1, chapter Introduction à la réalité virtuelle, pages 3–21. Presses de l'École des Mines de Paris, 3e edition, 2003.

[BB07] Aaron Bloomfield and Norman I. Badler. Collision Awareness Using Vibrotactile Arrays. In *Proceedings of the Virtual Reality Conference*, pages 163–170. IEEE Computer Society, March 2007.

[BBF+97] Steve Benford, John Bowers, Lennart E. Fahlén, Chris Greenhalgh, and Dave Snowdon. Embodiments, avatars, clones and agents for multi-user, multi-sensory virtual worlds. *Multimedia Systems*, 5(2) :93–104, 1997.

[BBK+01] Martin Bauer, Bernd Bruegge, Gudrun Klinker, Asa MacWilliams, Thomas Reicher, Stefan Riß, Christian Sandor, and Martin Wagner. Design of a component-based augmented reality framework. In *Proceedings of the IEEE and ACM International Symposium on Augmented Reality*, page 45, Washington, DC, USA, 2001. IEEE Computer Society.

[BD04] Marwan Badawi and Stéphane Donikian. Autonomous Agents Interacting With Their Virtual Environment Through Synoptic Objects. In *CASA'04 : Proceedings of Computer Animation and Social Agents*, 2004.

[BDDG03] Patrick Baudisch, Doug DeCarlo, Andrew T. Duchowski, and Wilson S. Geisler. Focusing on the Essential : Considering Attention in Display Design. *Communications of the ACM*, 46(3) :60–66, 2003.

[BGW06] Victor Bayon, Gareth Griffiths, and John R. Wilson. Multiple decoupled interaction : An interaction design approach for groupware interaction in co-located virtual environments. *International Journal of Human-Computer Studies*, 64(3) :192–206, 2006.

[BH97] Doug A. Bowman and Larry F. Hodges. An Evaluation of Techniques for Grabbing and Manipulating Remote Objects in Immersive Virtual Environments. In *I3D'97 : Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 35–38., New York, NY, USA, 1997. ACM Press.

[BJH+01] Allen Bierbaum, Christopher Just, Patrick Hartling, Kevin Meinert, Albert Baker, and Carolina Cruz-Neira. Vr juggler : A virtual platform for virtual reality application development. In *Proceedings of the Virtual Reality Conference*, page 89, Washington, DC, USA, 2001. IEEE Computer Society.

[BKLJP05] Doug A. Bowman, Ernst Kruijff, Joseph J. LaViola Jr., and Ivan Poupyrev. *3D User Interfaces : Theory and Practice*. Addison-Wesley/Pearson Education, 2005.

[BLO+05] Wolfgang Broll, Irma Lindt, Jan Ohlenburg, Iris Herbst, Michael Wittkämper, and Thomas Novotny. An Infrastructure for Realizing Custom-Tailored Augmented Reality User Interfaces. *IEEE Transactions on Visualization and Computer Graphics*, 11(6) :722–733, 2005.

[BM86] William Buxton and Brad A. Myers. A Study in Two-Handed Input. In *CHI'86 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 321–326, New York, NY, USA, 1986. ACM Press.

[BPO96] John Bowers, James Pycock, and Jon O'Brien. Talk and Embodiment in Collaborative Virtual Environments. In *CHI'96 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 58–65, 1996.

[CFH97] Lawrence D. Cutler, Bernd Fröhlich, and Pat Hanrahan. Two-Handed Direct Manipulation on the Responsive Workbench. In *I3D'97 : Proceedings of the 1997 symposium on Interactive 3D Graphics*, pages 107–114, New York, NY, USA, 1997. ACM Press.

[CH93] Christer Carlsson and Olof Hagsand. DIVE – a Multi-User Virtual Reality System. In *VRAIS'93 : Proceedings of the Virtual Reality Annual International Symposium*, pages 394–400. IEEE Computer Society, Sept 1993.

[CNSD+92] Carolina Cruz-Neira, Daniel J. Sandin, Thomas A. DeFanti, Robert V. Kenyon, and John C. Hart. The CAVE : audio visual experience automatic virtual environment. *Communications of the ACM*, 35(6) :64–72, 1992.

[DCDK98] Stéphane Donikian, Alain Chauffaut, Thierry Duval, and Richard Kulpa. GASP : From Modular Programming to Distributed Execution. In *Proceedings of the Computer Animation*, page 79, Washington, DC, USA, 1998. IEEE Computer Society.

[DDS+99] Cédric Dumas, Samuel Degrande, Grégory Saugis, Christophe Chaillou, Patricia Plénacoste, and Marie-Luce Viaud. Spin : a 3-D Interface for Cooperative Work. *Virtual Reality Society Journal*, May 1999.

[DeZ06] Thierry Duval and Chadi el Zammar. A Migration Mechanism to Manage Network Troubles while Interacting within Collaborative Virtual Environments. In *VRCIA'06 : Proceedings of the 2006 ACM international conference on Virtual Reality Continuum and Its Applications*, pages 417–420, New York, NY, USA, 2006. ACM Press.

[DF02] Thierry Duval and Aurélien Fenals. Faciliter la perception de l'interaction lors de manipulations coopératives simultanées en environnements virtuels 3d. In *IHM'02 : Annex of the Proceedings of Interaction Homme-Machine*, pages 29–32, Poitiers, France, November 2002.

[DF04] Pierre Dragicevic and Jean-Daniel Fekete. Support for Input Adaptability in the ICON Toolkit. In *ICMI'04 : Proceedings of the 6th international conference on Multimodal interfaces*, pages 212–219, New York, NY, USA, 2004. ACM.

[DF09] Thierry Duval and Cédric Fleury. An asymmetric 2D Pointer/3D Ray for 3D interaction within collaborative virtual environments. In *Web3D'09 : Proceedings of the 14th international conference on 3D Web technology*, pages 33–41, New York, NY, USA, 2009. ACM.

[DFNA08] Thierry Duval, Cédric Fleury, Bernard Nouailhas, and Laurent Aguerreche. Collaborative Exploration of 3D Scientific Data. In *VRST'08 : Proceedings of the 2008 ACM symposium on Virtual Reality Software and Technology*, pages 303–304, 2008.

[DHM02] Raimund Dachselt, Michael Hinz, and Klaus Meißner. CONTIGRA : An XML-Based Architecture for Component-Oriented 3D Applications. In *Web3D'02 : Proceedings of the seventh international conference on 3D Web technology*, pages 155–163, New York, NY, USA, 2002. ACM.

[DLB+05] Lionel Dominjon, Anatole Lécuyer, Jean-Marie Burkhardt, Paul Richard, and Simon Richir. Influence of control/display ratio on the perception of mass of manipulated objects in virtual environments. In *Proceedings of the Virtual Reality Conference*, pages 19–25. IEEE Computer Society, March 2005.

[DLT04] Thierry Duval and Christian Le Tenier. Interactions 3D coopératives en environnements virtuels avec OpenMASK pour l'exploitation d'objets techniques. *Mécanique & Industries*, 5 :129–137, 2004.

[DLT06] Thierry Duval, Anatole Lécuyer, and Sébastien Thomas. SkeweR : a 3D Interaction Technique for 2-User Collaborative Manipulation of Objects in Virtual Environments. In *3DUI'06 : Proceedings of the 3D User Interfaces*, pages 69–72. IEEE Computer Society, 2006.

[DM00] Thierry Duval and David Margery. Building objects and interactors for collaborative interactions with GASP. In *CVE'00 : Proceedings of the third international conference on Collaborative Virtual Environments*, pages 129–138, New York, NY, USA, 2000. ACM.

[DWM06a] Declan Delaney, Tomás Ward, and Seamus McLoone. On consistency and network latency in distributed interactive applications : a survey–Part I. *Presence : Teleoperators and Virtual Environments*, 15(2) :218–234, 2006.

[DWM06b] Declan Delaney, Tomás Ward, and Seamus McLoone. On Consistency and Network Latency in Distributed Interactive Applications : A Survey–Part II. *Presence : Teleoperators and Virtual Environments*, 15(4) :465–482, 2006.

[FBHH99] Mike Fraser, Steve Benford, Jon Hindmarsh, and Christian Heath. Supporting Awareness and Interaction through Collaborative Virtual Interfaces. In *UIST'99 : Proceedings of the 12th annual ACM symposium on User Interface Software and Technology*, pages 27–36. ACM Press, 1999.

[FDGA10] Cédric Fleury, Thierry Duval, Valérie Gouranton, and Bruno Arnaldi. Architectures and Mechanisms to Maintain efficiently Consistency in Collaborative Virtual Environments. In *SEARIS'10 : IEEE VR workshop on Software Engineering and Architectures for Realtime Interactive Systems*, Waltham, MA, USA, 2010.

[FGH02] Pablo Figueroa, Mark Green, and H. James Hoover. InTml : A Description Language for VR Applications. In *Web3D'02 : Proceedings of the seventh*

international conference on 3D Web technology, pages 53–58, New York, NY, USA, 2002. ACM.

[FGW01] Pablo Figueroa, Mark Green, and Benjamin Watson. A Framework for 3D Interaction Techniques. In *CAD/Graphics'2001*, volume 8, pages 22–24. International Academic Publishers, 2001.

[Fra95] John H. Frazer. *An evolutionary architecture.* Architectural Association, London, 1995.

[Fun95] Thomas A. Funkhouser. RING : a client-server system for multi-user virtual environments. In *I3D'95 : Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 85–92. ACM, 1995.

[GB95] Chris Greenhalgh and Steven Benford. MASSIVE : A Collaborative Virtual Environment for Teleconferencing. *ACM Transactions on Computer-Human Interaction*, 2(3) :239–261, 1995.

[GDGC06] S. Gaeremynck, S. Degrande, L. Grisoni, and C. Chaillou. Utilisation des composants pour la représentation des objets virtuels. In *Journées AFIG*, 2006.

[GG99] Carl Gutwin and Saul Greenberg. The effects of workspace awareness support on the usability of real-time distributed groupware. *ACM Transactions on Computer-Human Interaction*, 6(3) :243–281, 1999.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns : Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional, illustrated edition, November 1994.

[GlKP94] Rich Gossweiler, Robert J. laferriere, Michael L. Keller, and Randy Pausch. An Introductory Tutorial for Developing Multi-User Virtual Environments. *Presence : Teleoperators and Virtual Environments*, 3(4) :255–264, 1994.

[GMH04] Michael Gerhard, David Moore, and Dave Hobbs. Embodiment and copresence in collaborative interfaces. *International Journal of Human-Computer Studies*, 61(4) :453–480, 2004.

[GMMG08] Arturo S. García, José P. Molina, Diego Martínez, and Pascual González. Enhancing collaborative manipulation through the use of feedback and awareness in CVEs. In *VRCAI'08 : Proceedings of The 7th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry*, pages 1–5, New York, NY, USA, 2008. ACM.

[Han97] Chris Hand. A Survey of 3D Interaction Techniques. *Computer Graphics Forum*, 16(5) :269–281, 1997.

[HCK+99] Roger Hubbold, Jon Cook, Martin Keates, Simon Gibson, Toby Howard, Alan Murta, Adrian West, and Steve Pettifer. GNU/MAVERIK : a microkernel for large-scale virtual environments. In *VRST'99 : Proceedings of the ACM symposium on Virtual Reality Software and Technology*, pages 66–73, New York, NY, USA, 1999. ACM.

[HFH+98] Jon Hindmarsh, Mike Fraser, Christian Heath, Steve Benford, and Chris Greenhalgh. Fragmented Interaction : Establishing mutual orientation in virtual environments. In *CSCW'98 : Proceedings of the 1998 ACM Conference on Computer-Supported Cooperative Work*, pages 217–226. ACM Press, November 1998.

[HPGK94] Ken Hinckley, Randy Pausch, John C. Goble, and Neal F. Kassell. Passive real-world interface props for neurosurgical visualization. In *CHI'94 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 452–458, New York, NY, USA, 1994. ACM.

[HPPK98] Ken Hinckley, Randy Pausch, Dennis Proffitt, and Neal F. Kassell. Two-Handed Virtual Manipulation. *ACM Transactions on Computer-Human Interaction*, 5(3) :260–302, 1998.

[HRH06] David Hecht, Miriam Reiner, and Gad Halevy. Multimodal Virtual Environments : Response Times, Attention, and Presence. *Presence : Teleoperators and Virtual Environments*, 15(5) :515–523, 2006.

[HTP+97] Ken Hinckley, Joe Tullio, Randy Pausch, Dennis Proffitt, and Neal Kassell. Usability analysis of 3D rotation techniques. In *UIST'97 : Proceedings of the 10th annual ACM symposium on User Interface Software and Technology*, pages 1–10, New York, NY, USA, 1997. ACM.

[III99] Richard Rouse III. What's your perspective ? *SIGGRAPH Computer Graphics*, 33(3) :9–12, 1999.

[IIKK04] Hiroyasu Ichida, Yuichi Itoh, Yoshifumi Kitamura, and Fumio Kishino. Interactive Retrieval of 3D Virtual Shapes using Physical Objects. *Proceedings of the Virtual Reality Conference*, 0 :231, 2004.

[IMWB01] Brent Edward Insko, Michael J. Meehan, Mary C. Whitton, and Frederic P. Brooks. Passive haptics significantly enhances virtual environments. Technical report, The University of North Carolina at Chapel Hill, 2001.

[JJW+99] Sankar Jayaram, Uma Jayaram, Yong Wang, Hrishikesh Tirumali, Kevin Lyons, and Peter Hart. VADE : A Virtual Assembly Design Environment. *IEEE Computer Graphics and Applications*, 19(6) :44–50, 1999.

[KAKS02] John Kelso, Lance E. Arsenault, Ronald D. Kriz, and Steven G. Satterfield. DIVERSE : A Framework for Building Extensible and Reconfigurable Device Independent Virtual Environments. *Proceedings of the Virtual Reality Conference*, pages 183–190, 2002.

[Kal01] Marcelo Kallmann. *Object Interaction in Real-Time Virtual Environments*. PhD thesis, École Polytechnique Fédérale de Lausannne, 2001.

[KGH85] Myron W. Krueger, Thomas Gionfriddo, and Katrin Hinrichsen. VIDEOPLACE – An Artificial Reality. In *CHI'85 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 35–40, New York, NY, USA, 1985. ACM Press.

[KIKS00] Seahak Kim, Masahiro Ishii, Yasuharu Koike, and Makoto Sato. Development of Tension Based Haptic Interface and Possibility of its Application to Virtual Reality. In *VRST'00 : Proceedings of the ACM symposium on Virtual Reality Software and Technology*, pages 199–205. ACM, 2000.

[Kru93] Myron W. Krueger. Environmental Technology : Making the Real World Virtual. *Communications of the ACM*, 36(7) :36–37, 1993.

[KT98] Marcelo Kallmann and Daniel Thalmann. Modeling Objects for Interaction Tasks. In *Computer Animation and Simulation*, pages 73–86. Springer Computer Science, 1998.

[KTK+97] Kiyoshi Kiyokawa, Haruo Takemura, Yoshiaki Katayama, Hidehiko Iwasa, and Naokazu Yokoya. VLEGO : A Simple Two-handed Modeling Environment Based On Toy Block. In *VRST'97 : Proceedings of the ACM symposium on Virtual Reality Software and Technology*, pages 27–34, New York, NY, USA, 1997. ACM Press.

[KTK+98] Kiyoshi Kiyokawa, Haruo Takemura, Yoshiaki Katayama, Hidehiko Iwasa, and Naokazu Yokoya. VLEGO : A Simple Two-Handed 3D Modeler in a Virtual Environment. *Electronics and Communications in Japan, Part 3 : Fundamental Electronic Science*, 81(11) :18–28, November 1998.

[LCAA08] Xavier Larrodé, Benoît Chanclou, Laurent Aguerreche, and Bruno Arnaldi. OpenMASK : an Open-Source platform for Virtual Reality. In *SEARIS'08 : IEEE VR workshop on Software Engineering and Architectures for Realtime Interactive Systems*, Reno, NV, USA, 2008.

[LCK+00] Anatole Lécuyer, Sabine Coquillart, Abderrahmane Kheddar, Paul Richard, and Philippe Coiffet. Pseudo-Haptic Feedback : Can Isometric Input Devices Simulate Force Feedback ? *Proceedings of the Virtual Reality Conference*, pages 83–90, 2000.

[Lec09] Anatole Lecuyer. Simulating haptic feedback using vision : A survey of research and applications of pseudo-haptic feedback. *Presence : Teleoperators and Virtual Environments*, 18(1) :39–53, 2009.

[LPI07] Vincent LeClerc, Amanda Parkes, and Hiroshi Ishii. Senspectra : A computationally augmented physical modeling toolkit for sensing and visualization of structural strain. In *CHI'07 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 801–804, 2007.

[MAC+02] David Margery, Bruno Arnaldi, Alain Chauffaut, Stéphane Donikian, and Thierry Duval. OpenMASK : Multi-threaded | Modular Animation and Simulation Kernel | Kit : un bref survol. In *Proceedings of the Virtual Reality International Conference*. S. Richir, P. Richard, B. Taravel, juin 2002.

[MBS97] Mark R. Mine, Frederick P. Brooks, Jr., and Carlo H. Sequin. Moving objects in space : exploiting proprioception in virtual-environment interaction. pages 19–26, 1997.

[MC98] Katerina Mania and Alan Chalmers. A Classification for User Embodiment in Collaborative Virtual Environments. Technical report, University of Bristol, Bristol, UK, 1998.

[Min95] Mark R. Mine. Virtual Environment Interaction Techniques. Technical Report TR95-018, University of North Carolina, Department of Computer Science, 1995.

[MS93] Michael J. Massimino and Thomas B. Sheridan. Sensory Substitution for Force Feedback in Teleoperation. *Presence : Teleoperators and Virtual Environments*, 2(4) :344–352, 1993.

[MT95] Duncan C. Miller and Jack A. Thorpe. *SIMNET : The Advent of Simulator Networking*, volume 83, pages 1114–1123. IEEE, 1995.

[MZP+94] M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, and S. Zeswitz. NPSNET : A Network Software Architecture for Large-Scale Virtual Environment. *Presence : Teleoperators and Virtual Environments*, 3(4) :265–287, 1994.

[NM97] Haruo Noma and Tsutomu Miyasato. Cooperative Object Manipulation in Virtual Space using Virtual Physics. In *Proceedings of the ASME Dynamic Systems and Control Conference*, volume 61, pages 101–106, 1997.

[OF03] Alex Owal and Steven Feiner. The Flexible Pointer : An Interaction Technique for Selection in Augmented and Virtual Reality. In *Conference Supplement of UIST'03 (ACM Symposium on User Interface Software and Technology)*, pages 81–82, November 2003.

[OF04] Alex Olwal and Steven Feiner. Unit : Modular Development of Distributed Interaction Techniques for Highly Interactive User Interfaces. In *GRAPHITE'04 : Proceedings of the 2nd conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 131–138, New York, NY, USA, 2004. ACM.

[OHL+04] Jan Ohlenburg, Iris Herbst, Irma Lindt, Thorsten Fröhlich, and Wolfgang Broll. The MORGAN framework : enabling dynamic multi-user AR and VR projects. In *VRST'04 : Proceedings of the ACM symposium on Virtual Reality Software and Technology*, pages 166–169, New York, NY, USA, 2004. ACM.

[PBF02] Márcio S. Pinho, Doug A. Bowman, and Carla M.D.S Freitas. Cooperative Object Manipulation in Immersive Virtual Environments : Framework and Techniques. In *VRST'02 : Proceedings of the ACM symposium on Virtual Reality Software and Technology*, pages 171–178. ACM Press, 2002.

[PBWI96a] Ivan Poupyrev, Mark Billinghurst, Suzanne Weghorst, and Tadao Ichikawa. The Go-GO Interaction Technique for Direct Manipulation in VR. Unpublished SIGGRAPH Technical Sketch, 1996.

[PBWI96b] Ivan Poupyrev, Mark Billinghurst, Suzanne Weghorst, and Tadao Ichikawa. The Go-Go Interaction Technique : Non-Linear Mapping for Direct

Manipulation in VR. In *UIST'96 : Proceedings of the 9th annual ACM symposium on User Interface Software and Technology*, pages 79–80. ACM Press, 1996.

[PFC+97] Jeffrey S. Pierce, Andrew S. Forsberg, Matthew J. Conway, Seung Hong, Robert C. Zeleznik, and Mark R. Mine. Image Plane Interaction Techniques In 3D Immersive Environments. In *I3D'97 : Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 39–ff., New York, NY, USA, 1997. ACM Press.

[PLI06] Amanda Parkes, Vincent LeClerc, and Hiroshi Ishii. Glume : Exploring Materiality in a Soft Augmented Modular Modeling System. In *CHI'06 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1211–1216, 2006.

[PS96] Timothy Poston and Luis Serra. Dextrous Virtual Work. *Communications of the ACM*, 39(5) :37–45, 1996.

[PSP99] Jeffrey S. Pierce, Brian C. Stearns, and Randy Pausch. Voodoo dolls : seamless interaction at multiple scales in virtual environments. In *I3D'99 : Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 141–145, New York, NY, USA, 1999. ACM Press.

[PWBI98] Ivan Poupyrev, Suzanne Weghorst, Mark Billinghurst, and Tadao Ichikawa. Egocentric Object Manipulation in Virtual Environments : Empirical Evaluation of Interaction Techniques. *Computer Graphics Forum*, 17(3), 1998.

[RHWF06] Kai Riege, Thorsten Holtkamper, Gerold Wesche, and Bernd Frohlich. The Bent Pick Ray : An Extended Pointing Technique for Multi-User Interaction. In *3DUI'06 : Proceedings of the 3D User Interfaces*, pages 62–65, Washington, DC, USA, 2006. IEEE Computer Society.

[RS01] Gerhard Reitmayr and Dieter Schmalstieg. An Open Software Architecture for Virtual Reality Interaction. In *VRST'01 : Proceedings of the ACM symposium on Virtual Reality Software and Technology*, pages 47–54, New York, NY, USA, 2001. ACM.

[RSJ02] Roy A. Ruddle, Justin C. D. Savage, and Dylan M. Jones. Symmetric and asymmetric action integration during cooperative object manipulation in virtual environments. *ACM Transactions on Computer-Human Interaction*, 9(4) :285–308, 2002.

[RWOS03a] David Roberts, Robin Wolff, Oliver Otto, and Anthony Steed. Constructing a Gazebo : supporting teamwork in a tightly coupled, distributed task in virtual reality. *Presence : Teleoperators and Virtual Environments*, 12(6) :644–657, 2003.

[RWOS03b] David Roberts, Robin Wolff, Oliver Otto, and Anthony Steed. Constructing a Gazebo : Supporting Teamwork in a Tightly Coupled, Distributed Task in Virtual Reality. *Presence : Teleoperation and Virtual Environments*, 12(6) :644–657, 2003.

[SBLG+07] Jean Sreng, Florian Bergez, Jérémie Le Garrec, Anatole Lécuyer, and Claude Andriot. Using an Event-Based Approach to Improve the Multimodal Rendering of 6DOF Virtual Contact. In *VRST'07 : Proceedings of the ACM International Symposium on Virtual Reality Software and Technology*, pages 165–173, New York, NY, USA, 2007. ACM.

[SCP95] Richard Stoakley, Matthew J. Conway, and Randy Pausch. Virtual Reality on a WIM : Interactive Worlds in Miniature. In *CHI'95 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 265–272. ACM Press/Addison-Wesley Publishing Co., 1995.

[SE00] Ben Salem and Nic Earle. Designing a Non-Verbal Language for Expressive Avatars. In *CVE'00 : Proceedings of the third international conference on Collaborative Virtual Environments*, pages 93–101, New York, NY, USA, 2000. ACM Press.

[SG94] Chris Shaw and Mark Green. Two-Handed Polygonal Surface Design. In *UIST'94 : Proceedings of the 7th annual ACM symposium on User Interface Software and Technology*, pages 205–212, New York, NY, USA, 1994. ACM Press.

[SGLS93] Chris Shaw, Mark Green, Jiandong Liang, and Yunqi Sun. Decoupled simulation in virtual reality with the MR toolkit. *Transactions on Informations Systems*, 11(3) :287–317, 1993.

[Sho85] Ken Shoemake. Animating rotation with quaternion curves. *ACM SIGGRAPH Computer Graphics*, 19(3) :245–254, 1985.

[SJF09] H. Salzmann, J. Jacobs, and B. Froehlich. Collaborative Interaction in Co-Located Two-User Scenarios. In *Proceedings of JVRC 2009 (Joint Virtual Reality Conference of EGVE - ICAT - EuroVR 2009)*, pages 85–92, 2009.

[SLMA06] Jean Sreng, Anatole Lécuyer, Christine Mégard, and Claude Andriot. Using Visual Cues of Contact to Improve Interactive Manipulation of Virtual Objects in Industrial Assembly/Maintenance Simulations. *IEEE Transactions on Visualization and Computer Graphics*, 12(5) :1013–1020, 2006.

[SRS91] Emanuel Sachs, Andrew Roberts, and David Stoops. 3-Draw : A Tool for Designing 3D Shapes. *Computer Graphics and Applications, IEEE*, 11(6) :18–26, 1991.

[STP08] Ross T. Smith, Bruce H. Thomas, and Wayne Piekarski. Digital Foam Interaction Techniques for 3D Modeling. In *VRST'08 : Proceedings of the 2008 ACM symposium on Virtual Reality Software and Technology*, pages 61–68, 2008.

[SVP07] Pedro Sequeira, Marco Vala, and Ana Paiva. What can I do with this ? : Finding possible interactions between characters and objects. In *AAMAS'07 : Proceedings of the 6th international joint conference on Autonomous Agents and Multiagent Systems*, pages 1–7. ACM, 2007.

[THS+01] Russell M. Taylor, II, Thomas C. Hudson, Adam Seeger, Hans Weber, Jeffrey Juliano, and Aron T. Helser. VRPN : a device-independent, network-transparent VR peripheral system. In *VRST'01 : Proceedings of the ACM symposium on Virtual Reality Software and Technology*, pages 55–61, New York, NY, USA, 2001. ACM.

[UI00] Brygg Ullmer and Hiroshi Ishii. Emerging Frameworks for Tangible User Interfaces. *IBM Systems Journal*, 39(3-4) :915–931, 2000.

[WIA+04] Ryoichi Watanabe, Yuichi Itoh, Masatsugu Asai, Yoshifumi Kitamura, Fumio Kishino, and Hideo Kikuchi. The Soul of ActiveCube — Implementing a Flexible, Multimodal, Three-Dimensional Spatial Tangible Interface. *Computers in Entertainment*, 2(4) :15–15, 2004.

[WR99] Colin Ware and Jeff Rose. Rotating virtual objects with real handles. *ACM Transactions on Computer-Human Interaction*, 6(2) :162–180, 1999.

[ZSF05] Ying Zhang, Reza Sotudeh, and Terrence Fernando. The Use of Visual and Auditory Feedback for Assembly Task Performance in a Virtual Environment. In *SCCG'05 : Proceedings of the 21st spring conference on Computer graphics*, pages 59–66, New York, NY, USA, 2005. ACM Press.

Webographie

[ART] Site de A.R.T. GmbH. http://www.ar-tracking.de/.

[Bul] Site de Bullet. http://www.bulletphysics.com/.

[COL] Site de COLLADA. https://www.collada.org/.

[COR] Site de CORBA. http://www.omg.org/.

[Hav] Site de Havok. http://www.havok.com/.

[Ogr] Site de Ogre3D. http://www.ogre3d.org/.

[Par] Site du projet ANR Part@ge. http://www.rntl-partage.fr/.

[per] Site du projet RNTL Perf-rv. http://www.perfrv.org/.

[Phy] Site de NVIDIA PhysX. http://developer.nvidia.com/object/physx.html.

[Sur] Site de Surface de microsoft. http://www.microsoft.com/surface/.

[Vir] Site de Virtools. http://www.3ds.com/products/3dvia/3dvia-virtools/.

[VRM] Site de VRML 97. http://www.web3d.org/x3d/specifications/vrml/.

[VRP] Site de VRPN. http://www.cs.unc.edu/Research/vrpn.

[X3D] Site de X3D. http://www.web3d.org/x3d/.

[XML] Site de XML (Extensible Markup Language). http://www.w3.org/XML/.

Table des figures

INSA de Rennes
Service des formations

AVIS DU JURY SUR LA REPRODUCTION DE LA THESE SOUTENUE

Titre de la thèse : Partage d'interactions en environnements virtuels : de nouvelles techniques collaboratives basées sur un protocole de dialogue générique

Nom Prénom de l'auteur : AGUERRECHE Laurent

Membres du jury :

Madame	COQUILLART
Madame	THOUVENIN
Monsieur	ARNALDI
Monsieur	DUVAL
Monsieur	HACHET
Monsieur	PAZAT

Président du jury :

Date de la soutenance : 04/06/2010

Reproduction de la thèse soutenue :

- ☒ Thèse pouvant être reproduite en l'état
- ☐ Thèse ne pouvant être reproduite
- ☐ Thèse pouvant être reproduite après corrections suggérées

Le Directeur,

A. JIGOREL

Rennes, le 04/06/2010

Signature du Président du jury

Résumé

La réalité virtuelle permet à des utilisateurs de manipuler des objets virtuels depuis un même lieu ou des sites géographiques distants. Toutefois, un objet virtuel ne peut souvent être manipulé que par une seule personne à la fois.

Dans cette thèse, nous définissons un protocole d'interaction décrivant le dialogue entre des outils d'interaction et des objets interactifs. Ce protocole basé messages gère les interactions locales ou distantes sur des objets interactifs en mono et multi-utilisateurs. Il inclut également une gestion des permissions pour les accès des outils d'interaction. Par ailleurs, le développement de ce protocole a permis de définir ce que sont des outils d'interaction et des objets interactifs. Cette caractérisation a donné lieu à une implémentation reposant sur un ensemble de composants logiciels réutilisables qui permettent notamment à un outil d'interaction de modifier la méthode de sélection et de modification d'objets interactifs, et à un objet interactif de modifier son comportement lorsqu'il est manipulé. De plus, cette caractérisation a permis de proposer des extensions du format COLLADA afin d'y incorporer une gestion d'interactions collaboratives.

Une nouvelle technique d'interaction multi-utilisateurs est proposée. Elle déduit des mouvements selon 6 degrés de liberté à partir des positions (sans les orientations) de 3 mains virtuelles. Cette technique permet donc à deux ou trois utilisateurs de manipuler ensemble un objet virtuel. La possibilité d'un positionnement précis des mains virtuelles d'un utilisateur lui permet de mieux imiter la réalité. En effet, selon les positionnements des mains, le ressenti de l'utilisateur lors de la manipulation d'un objet, en particulier des objets encombrants, peut varier. Cette technique cherche donc à obtenir des placements des mains et des gestes réalistes chez l'utilisateur. Son implémentation a été réalisée au dessus du protocole d'interaction.

Un nouveau concept d'interface tangible reconfigurable pour des interactions en mono ou multi-utilisateurs est également proposée. Cette interface constitue une solution pour la manipulation d'objets virtuels 3D en approximant leur forme. En effet, l'interface tangible reconfigurable, nommée RTD, propose un maillage physique reconfigurable constitué de points de manipulation à placer sur un objet virtuel, tels des points d'accroche. Ce maillage de points esquisse ainsi l'objet à manipuler. Deux sortes de RTD sont illustrées dans cette thèse. Une première sorte est de forme triangulaire. Elle est nommée RTD-3 et comporte donc trois points de manipulation. Les expérimentations effectuées montrent une nette préférence des utilisateurs envers l'usage de cette technique plutôt que celui des techniques classiques de la réalité virtuelle telles que la moyenne de mouvements et la séparation des degrés de liberté selon les utilisateurs. Une deuxième sorte de RTD comporte quatre points de manipulation et se trouve sous deux versions. Le RTD-4 plan place ses quatre points sur un plan et permet d'obtenir un quadrilatère. Le RTD-4 non-plan dispose en ses quatre points de manipulation d'articulations permettant d'obtenir un tétraèdre. Une forme 3D pour le RTD devrait conduire à une meilleure correspondance entre la forme du RTD et la forme de l'objet virtuel à manipuler.

Abstract

Virtual reality enables users to manipulate virtual objects from local or distant sites. Nevertheless, virtual objects are usually manipulated by only one user at a time.

In this thesis, we define an interaction protocol describing how interaction tools and interactive objects communicate. This message-based protocol handles local and distant interactions on interactive objects for single or multi-user situations. It also manages permissions for access of interaction tools. Moreover the development of this protocol led to the definition of interaction tools and interactive objects. This characterization has driven an implementation based on a set of reusable software components that let an interaction tool modify how it selects or modifies interactive objects, and let an interactive object modify its own behaviour when it is manipulated. On top of that, extensions to the COLLADA file format based on this characterization and dedicated to collaborative interaction are proposed.

A new technique for multi-user interaction is proposed. Movements in 6 degrees of freedom are deduced from the positions (not orientations) provided by 3 virtual hands. This technique thus enables two or three users to manipulate together virtual objects. The ability given to users to put precisely their virtual hands brings them a way to imitate better reality. In fact, users' feeling during a manipulation can depend on how users hold this object notably with cumbersome objects. Therefore, this technique aims to obtain realistic hand positions and gestures of users. The implementation is built upon the interaction protocol.

A new concept of reconfigurable tangible interface for single or multi-user interaction is also proposed. This interface provides a way to manipulate virtual 3D objects. This interface, named RTD, proposes a reconfigurable physical mesh made up of manipulation points that have to be put on a virtual object, like grasping points. This mesh thus roughly sketches the form of the virtual object to manipulate. Two kinds of RTDs are illustrated in this thesis. A first kind of RTD is a triangular shape. It is named RTD-3 and so made up of three manipulation points. Experiments show that users prefer this technique over classical techniques of virtual reality like the mean of users' movements or the separation of degrees of freedom. A second kind of RTD is made up of four manipulation points and exists in two versions. The plane RTD-4 has its four points laying on a plane to create a quadrilateral. The non-plane RTD-4 uses articulated joints at its manipulation points to obtain a tetrahedron. A 3D form for a RTD would lead to better matching between the form of a RTD and the form of the virtual object to manipulate.

N°d'ordre : D10-07

Institut National des Sciences Appliquées de Rennes
20, Avenue des Buttes de Coësmes . CS 14315 . F-35043 Rennes Cedex
Tel : 02 23 23 82 00 – Fax : 02 23 23 83 96